```haskell
import System.IO

-- 1

{-

Y E = E (Y E)

LET gef = \ f x y . (iszero x ? iszero y : (iszero y ? True : f (prev x) (f prev y)))

let GE = Y gef

-}

-- 2

data SExpr
  = EVal Integer
  | EMul SExpr SExpr
  | ESub SExpr SExpr
  deriving (Show, Eq)

evalSE :: SExpr -> Integer
evalSE (EVal i) = i
evalSE (EMul e1 e2) = evalSE e1 * evalSE e2
evalSE (ESub e1 e2) = evalSE e1 - evalSE e2


-- 3

{-

(++) [] ys = ys                                  -- 1
(++) (x:xs) ys = x : (xs++ys)                    -- 2

map f [] = []                                    -- 5
map f (x:xs) = f x : map f xs                    -- 6


map f (xs ++ ys) = map f xs ++ map f ys

xs = []

L = map f ([] ++ ys) =|1
  = map f ys
P = map f [] ++ map f ys =|5
  = [] ++ map f ys =|1
  = map f ys
L = P


I.P.: map f (as ++ ys) = map f as ++ map f ys

xs = (a:as)

map f ((a:as) ++ ys) = map f (a:as) ++ map f ys

L = map f ((a:as) ++ ys) =|2
  = map f (a : (as++ys)) =|6
  = f a : map f (as ++ ys) =|IP
  = f a : (map f as ++ map f ys)
P = map f (a:as) ++ map f ys =|6
  = (f a : map f as) ++ map f ys =|2
  = f a : (map f as ++ map f ys)

L=P

Q.E.D.
-}


-- 4

type ProcName
  = String

type ProcId
  = Integer

type ProcTimeS
  = Integer

data ProcLog
  = Item ProcName ProcId ProcTimeS
  deriving (Show,Eq)

fileRead :: String -> IO ([ProcLog])
fileRead f = do
  h <- openFile f ReadMode
  c <- hGetContents h
  let log = parseLines $ lines c
  (return $! log) >>= (\l -> hClose h >> return l)

parseLine :: String -> ProcLog
parseLine l =
  Item name ((read sid)::Integer) (((read hh)::Integer) * 3600
    + ((read mm)::Integer)*60 + (read ss)::Integer)
  where
    (name,r0) = span (/= ' ') l
    (sid,r1)  = span (/= ' ') $ dropWhile (== ' ') r0
    time      = dropWhile (== ' ') r1
    (hh,r2)   = span (/= ':') time
    (mm,r3)   = span (/= ':') $ tail r2
    ss        = tail r3

parseLines :: [String] -> [ProcLog]
parseLines (l:ls) = parseLine l : parseLines ls
parseLines [] = []

-- EOF
```