Lightweight Language Weekend 発表資料 「君ならどう書く」 お題: ls-lR Shell 言語: Haskell 発表者: 花谷

■ Main.hs

```
 1 module Main where
 2
 3 import List
 4 import IO
 5 import System
 6 import Tree
 7 import FileStat
 8 import FindExpr
 9
10 type DTree = Tree FileStat
11
12 main :: IO ()
13 main = getArgs >>= flip openFile ReadMode . head >>= hGetContents
14        >>= repl . (:[]) . makeDirectoryTree
15
16 repl :: [DTree] -> IO ()
17 repl trees = do putStr "ls-lR> "
18                 hFlush stdout
19                 cmd <- getLine `catch` (¥e -> return "quit")
20                 case words cmd of
21                   ("quit":    _) -> return ()
22                   ("pwd" :    _) -> cmdPwd      trees      >>= repl
23                   ("ls"  :    _) -> cmdLs       trees      >>= repl
24                   ("cd"  : rest) -> cmdCd       trees rest >>= repl
25                   ("dfs" : rest) -> cmdFind dfs trees rest >>= repl
26                   ("bfs" : rest) -> cmdFind bfs trees rest >>= repl
27                   _ -> putStrLn "Command not found" >> repl trees
28
29 cmdPwd :: [DTree] -> IO [DTree]
30 cmdPwd ts = (putStrLn . ("/" ++) . path) ts >> return ts
31
32 cmdLs :: [DTree] -> IO [DTree]
33 cmdLs ts@((Branch _ xs):_) = mapM_ (putStrLn . rawString . node) xs >> return ts
34
35 cmdCd :: [DTree] -> [String] -> IO [DTree]
36 cmdCd ts [] = return [last ts]
37 cmdCd ts (arg:_)
38     = case tracePath ts arg of
39         (Leaf a):_ -> putStrLn "Not a directory" >> return ts
40         []         -> putStrLn "No such file or directory" >> return ts
41         xs         -> return xs
42
43 cmdFind :: ([DTree] -> [[DTree]]) -> [DTree] -> [String] -> IO [DTree]
44 cmdFind f tts@(t:ts) xs
45     = case parseExpr xs of
46         Just expr -> mapM_ putStrLn [ "./" ++ path e
47                                     | e <- tail $ f [t]
48                                     , match expr (node $ head e)
49                                     ] >> return tts
50         Nothing   -> putStrLn "Unknown expression" >> return tts
51
52 path :: [DTree] -> String
53 path = concat . intersperse "/" . map (name . node) . tail . reverse
54
55 tracePath :: [DTree] -> String -> [DTree]
56 tracePath ts str
57     | head str == '/' = tracePath' [last ts] $ splitPath $ tail str
58     | otherwise       = tracePath' ts $ splitPath str
59   where
60     tracePath' :: [DTree] -> [String] -> [DTree]
61     tracePath' ts [] = ts
62     tracePath' (t@(Branch _ xs):ts) (p:ps)
63         | p == "."  = tracePath' (t:ts) ps
64         | p == ".." = tracePath' (if null ts then [t] else ts) ps
65         | otherwise = case find ((p==) . name . node) xs of
66                         Just x -> tracePath' (x:t:ts) ps
67                         Nothing -> []
68
69 makeDirectoryTree :: String -> DTree
70 makeDirectoryTree
71     = foldl insert (Branch (FS {name = "."}) []) . map parse . parags . lines
72   where
73     parse (x:_:xs) = (tail $ splitPath $ init x, map makeNode xs)
74     makeNode x = let fs = fileStat x in
75                  if isDirectory fs then Branch fs [] else Leaf fs
```