

# psilo

This is the main program outline. If an argument is present on the command line then we execute the program in that file and halt. Otherwise we fire up a repl.

```
module Main where

import Parser
import Syntax
import Evaluator

import Control.Monad.Trans
import System.Console.Haskeline

import System.Environment
import System.IO
import Text.Parsec
```

`eval` amounts to taking a line of code (a line in the repl, an expression otherwise), getting the `Expr` value from the parser and then running a `Machine` with said value.

The result is the state of the machine after it has been run.

```
eval :: Either ParseError [Expr ()] -> MStore -> IO [MStore]
eval res store = do
  case res of
    Left err -> print err >> return [store]
    Right ex -> mapM execute (ex :: [Expr ()]) >>= return

  where execute v = do
    (val, store') <- (runMachineWithState store ev') . interpret $ v
    putStrLn . show $ val
    return store'
    ev' = case (mFinalEnv store) of
      Nothing -> initialEnv
      Just e -> MEnv e
```

The repl is nothing more than calling `eval` in an endless loop.

```
repl :: IO ()
repl = runInputT defaultSettings (loop initialState) where
  loop store = do
    minput <- getInputLine "psilo> "
    case minput of
      Nothing -> outputStrLn "Goodbye."
      Just input -> do
        case input of
          ":state" -> liftIO (putStrLn . show $ store) >> loop store
          -         -> do
            (store':_) <- liftIO $ eval (parseTopLevel input) store
            loop store'
```

If we gave psilo a file name we parse and evaluate it instead of starting the REPL.

```
execFile :: String -> IO ()
execFile fname = do
  parsed <- liftIO $ parseFile fname
  eval parsed initialState
  return ()

main :: IO ()
main = do
  args <- getArgs
  case args of
    [] -> repl >> return ()
    [fname] -> execFile fname >> return ()
```