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
```

`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 :: String -> MStore -> IO MStore
eval line store = do
    let res = parseTopLevel line
    case res of
        Left err -> print err >> return store
        Right [ex] -> execute (ex :: Expr ()) >>= return

    where execute v = do
              (val, store') <- runMachine . interpret $ v
              putStrLn . show $ val
              return store'
```

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

```
repl :: IO ()
repl = runInputT defaultSettings (loop initialStore) 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 input store
                        loop store'
```

If we execute `eval` on the contents of a file of code, we have a normal interpreter:

```
execFile :: String -> IO ()
execFile fname = readFile fname >>= (flip eval) initialStore >> return ()

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