```
{-# LANGUAGE OverloadedStrings #-}
{-
Copyright (C) 2012-2024 John MacFarlane <jgm@berkeley.edu>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
-}
import Text.Pandoc
import Text.Pandoc.MIME
import Control.DeepSeq (force)
import Control.Monad.Except (throwError)
import qualified Text.Pandoc.UTF8 as UTF8
import qualified Data.ByteString as B
import qualified Data.Text as T
import Test.Tasty.Bench
-- import Gauge
import qualified Data.ByteString.Lazy as BL
import Data.Maybe (mapMaybe)
import Data.List (sortOn)
import Text.Pandoc.Format (FlavoredFormat(..))

readerBench :: Pandoc
            -> T.Text
            -> Maybe Benchmark
readerBench _ name
  | name `elem` ["bibtex", "biblatex", "csljson"] = Nothing
readerBench doc name = either (const Nothing) Just $
  runPure $ do
    (rdr, rexts) <- getReader $ FlavoredFormat name mempty
    (wtr, wexts) <- getWriter $ FlavoredFormat name mempty
    tmpl <- Just <$> compileDefaultTemplate name
    case (rdr, wtr) of
      (TextReader r, TextWriter w) -> do
        inp <- w def{ writerWrapText = WrapAuto
                    , writerExtensions = wexts
                    , writerTemplate = tmpl } doc
        return $ bench (T.unpack name) $
          nf (either (error . show) id . runPure . r def) inp
      (ByteStringReader r, ByteStringWriter w) -> do
        inp <- w def{ writerWrapText = WrapAuto
                    , writerExtensions = wexts
                    , writerTemplate = tmpl } doc
        return $ bench (T.unpack name) $
          nf (either (error . show) id .
                runPure . r def{readerExtensions = rexts}) inp
      _ -> throwError $ PandocSomeError $ "text/bytestring format mismatch: "
                           <> name

getImages :: IO [(FilePath, MimeType, BL.ByteString)]
getImages = do
  ll <- B.readFile "test/lalune.jpg"
  mv <- B.readFile "test/movie.jpg"
  return [("lalune.jpg", "image/jpg", BL.fromStrict ll)
         ,("movie.jpg", "image/jpg", BL.fromStrict mv)]

writerBench :: [(FilePath, MimeType, BL.ByteString)]
```

```
            -> Pandoc
            -> T.Text
            -> Maybe Benchmark
writerBench _ _ name
  | name `elem` ["bibtex", "biblatex", "csljson"] = Nothing
writerBench imgs doc name = either (const Nothing) Just $
  runPure $ do
    (wtr, wexts) <- getWriter $ FlavoredFormat name mempty
    case wtr of
      TextWriter writerFun ->
        return $ bench (T.unpack name)
               $ nf (\d -> either (error . show) id $
                       runPure $ do
                         mapM_ (\(fp,mt,bs) -> insertMedia fp (Just mt) bs) imgs
                         writerFun def{ writerExtensions = wexts} d)
                    doc
      ByteStringWriter writerFun ->
        return $ bench (T.unpack name)
               $ nf (\d -> either (error . show) id $
                       runPure $ do
                         mapM_ (\(fp,mt,bs) -> insertMedia fp (Just mt) bs) imgs
                         writerFun def{ writerExtensions = wexts} d)
                    doc

main :: IO ()
main = do
  inp <- UTF8.toText <$> B.readFile "test/testsuite.txt"
  let opts = def
  let doc = either (error . show) force $ runPure $ readMarkdown opts inp
  defaultMain
    [ env getImages $ \imgs ->
      bgroup "writers" $ mapMaybe (writerBench imgs doc . fst)
                         (sortOn fst
                           writers :: [(T.Text, Writer PandocPure)])
    , bgroup "readers" $ mapMaybe (readerBench doc . fst)
                         (sortOn fst
                           readers :: [(T.Text, Reader PandocPure)])
    ]
```