yaml: Parsing and rendering YAML

Section exercise: write a program using the `rio` template that receives a database connection string as an environment variable, by using a YAML config file.

- High level API based on aeson
- Low-level, streaming API (we won't cover it)
- Config file helpers
- Leverages C libyaml for functionality

Basic usage

```haskell
#!/usr/bin/env stack
-- stack --resolver lts-12.21 script
{-# LANGUAGE RecordWildCards #-}
{-# LANGUAGE OverloadedStrings #-}
import Data.Aeson (withObject) -- should be provided by yaml...
import Data.Text (Text)
import Data.Vector (Vector)
import Data.Yaml

data Person = Person
  { personName :: !Text
  , personAge :: !Int
  }
  deriving (Show, Eq)
-- Could use Generic deriving, doing it by hand
instance ToJSON Person where
  toJSON Person {..} = object
    [ "name" .= personName
    , "age"  .= personAge
    ]
instance FromJSON Person where
  parseJSON = withObject "Person" $ \o -> Person
    <$> o .: "name"
    <*> o .: "age"

main :: IO ()
main = do
  let bs = encode
        [ Person "Alice" 25
        , Person "Bob" 30
        , Person "Charlie" 35
        ]
  people <-
    case decodeEither' bs of
      Left exc -> error $ "Could not parse: " ++ show exc
      Right people -> return people

  let fp = "people.yaml"
  encodeFile fp (people :: Vector Person)
  res <- decodeFileEither fp
  case res of
    Left exc -> error $ "Could not parse file: " ++ show exc
    Right people2
      | people == people2 -> mapM_ print people
      | otherwise -> error "Mismatch!"
```

- Encode/decode to both `ByteString` and files
- Prefer the explicit exception functions
    - 20/20 hindsight: they'd be the default

Config files

- Common use case for YAML
- Would be nice to allow for env var override
- This is the default Yesod scaffolding config approach
- Stolen from my Yesod talk, apologies :)

YAML file itself

```
aws-secret: _env:AWS_SECRET
home-response: _env:HOME_RESPONSE:Hello World
```

- Special syntax in YAML to allow env overriding
- aws-secret: must have an env var
- home-response: optional

Haskell code

```
#!/usr/bin/env stack
-- stack --resolver lts-12.21 script
{-# LANGUAGE OverloadedStrings #-}
import Data.Aeson (withObject)
import Data.Text (Text)
import Data.Yaml
import Data.Yaml.Config

data Config = Config
  { awsSecret    :: !Text
  , homeResponse :: !Text
  }
  deriving Show
instance FromJSON Config where
  parseJSON = withObject "Config" $ \o -> Config
    <$> o .: "aws-secret"
    <*> o .: "home-response"

main :: IO ()
main = do
  config <- loadYamlSettingsArgs [] useEnv
  print (config :: Config)
```

- `FromJSON`: normal aeson/yaml code
- No `ToJSON` needed
- Get config file name from command line arguments
- Use environment variables when available

Usage

```
$ ./Main.hs
Main.hs: loadYamlSettings: No configuration provided
$ ./Main.hs config.yaml
Main.hs: Could not convert to AppSettings: expected Text,
         encountered Null
$ AWS_SECRET=foobar ./Main.hs config.yaml
Config {awsSecret = "foobar", homeResponse = "Hello World"}
$ AWS_SECRET=foobar HOME_RESPONSE=Goodbye ./Main.hs config.yaml
Config {awsSecret = "foobar", homeResponse = "Goodbye"}
```

- Must provide config file(s) on command line
- Must provide `AWS_SECRET`
- If provided, `HOME_RESPONSE` changes that response payload

Exercises

Write a Haskell program to generate the following YAML file:

```yaml
- title: Star Wars
  director: George Lucas
- title: Transformers
  director: Michael Bay
```

Write a Haskell program to convert a JSON formatted file to YAML format, and vice-versa.