```haskell
{-# LANGUAGE OverloadedStrings #-}
{-# LANGUAGE RecordWildCards   #-}

module OpenData
    ( Item(..)
    , ItemType(..)
    , decodeItems
    , decodeItemsFromFile
    , encodeItems
    , encodeItemsToFile
    , filtrerCountryItems
    , itemHeader
    , japanItem
    , japanRecord
    )
        where

-- base
import Control.Exception (IOException)
import qualified Control.Exception as Exception
import qualified Data.Foldable as Foldable
-- bytestring
import Data.ByteString.Lazy (ByteString)
import qualified Data.ByteString.Lazy as ByteString
-- cassava
import Data.Csv
  ( DefaultOrdered(headerOrder)
  , FromField(parseField)
  , FromNamedRecord(parseNamedRecord)
  , Header
  , ToField(toField)
  , ToNamedRecord(toNamedRecord)
  , (.:)
  , (.=)
  )
import qualified Data.Csv as Cassava
-- text
import Data.Text (Text)
import qualified Data.Text.Encoding as Text
-- vector
import Data.Vector (Vector)
import qualified Data.Vector as Vector

data Item =
  Item
    { itemName :: Text
    , itemLink :: Text
    , itemType :: ItemType
    }
  deriving (Eq, Show)

data ItemType
  = Country
  | Other Text
  deriving (Eq, Show)

japanRecord :: ByteString
japanRecord =
    "Japan,https://www.data.go.jp/?lang=english,International Country"

japanItem :: Item
japanItem =
    Item
        { itemName = "Japan"
        , itemLink = "http://www.data.go.jp/"
        , itemType = Country
        }

instance FromNamedRecord Item where
    parseNamedRecord m =
        Item
        <$> fmap Text.decodeLatin1 (m .: "Item")
        <*> m .: "Link"
        <*> m .: "Type"

instance FromField ItemType where
    parseField "International Country" =
        pure Country
    parseField otherType =
        Other <$> parseField otherType

instance ToNamedRecord Item where
    toNamedRecord Item{..} =
        Cassava.namedRecord
        [ "Item" .= itemName
        , "Link" .= itemLink
        , "Type" .= itemType
        ]

instance ToField ItemType where
    toField Country =
        "International Country"

    toField (Other otherType) =
        toField otherType

instance DefaultOrdered Item where
    headerOrder _ =
        Cassava.header
       [ "Item"
        , "Link"
        , "Type"
        ]

decodeItems
    :: ByteString
    -> Either String (Vector Item)
decodeItems =
    fmap snd . Cassava.decodeByName

decodeItemsFromFile
    :: FilePath
    -> IO (Either String (Vector Item))
decodeItemsFromFile filePath =
        catchShowIO (ByteString.readFile filePath)
        >>= return . either Left decodeItems

catchShowIO
    :: IO a
    -> IO (Either String a)
catchShowIO action =
    fmap Right action `Exception.catch` handleIOException
        where
            handleIOException
                :: IOException
                -> IO (Either String a)
            handleIOException =
                return . Left . show


encodeItems
    :: Vector Item
    -> ByteString
encodeItems =
    Cassava.encodeDefaultOrderedByName . Foldable.toList

encodeItemsToFile
    :: FilePath
    -> Vector Item
    -> IO (Either String ())
encodeItemsToFile filePath =
    catchShowIO . ByteString.writeFile filePath . encodeItems

filtrerCountryItems
    :: Vector Item
    -> Vector Item
```

```
filtrerCountryItems =
    Vector.filter isCountryItem

isCountryItem
    :: Item
    -> Bool
isCountryItem =
    (==) Country . itemType

itemHeader :: Header
itemHeader =
    Vector.fromList
    [ "Item"
    , "Link"
    , "Type"
    ]
```