Manejo de ficheros

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Lectura de ficheros con `readFile`

- Supongamos un fichero de texto, `Ejemplo_1.txt`, con el siguiente contenido:

```
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.
```

- Podemos leer el contenido completo del archivo con `readFile`:

```
Prelude> :type readFile
readFile :: FilePath -> IO String

Prelude> readFile "Ejemplo_1.txt"
"Este fichero tiene tres lineas\nesta es la segunda y\nesta la tercera.\n"
```

- Como vemos, nos devuelve todo en una cadena, incluyendo los salto de línea correspondientes (`\n`).

Lectura de ficheros con `readFile`

Lanzando desde intérprete

Podemos estructurarlo, lanzarlo y visualizar el resultado de varias formas:

- Lanzando desde intérprete:

```
Prelude> readFile "Ejemplo_1.txt"
"Este fichero tiene tres lineas\nesta es la segunda y\nesta la tercera.\n"

Prelude> cs <- readFile "Ejemplo_1.txt"
Prelude> putStrLn cs
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.
```

Lectura de ficheros con `readFile`

- Alternativas de funciones para leer ficheros:
  A. Una función que muestre directamente el contenido del fichero:

```
muestraContenidoFichero :: FilePath -> IO ()
muestraContenidoFichero f = do
    cs <- readFile f
    putStrLn cs
```

  B. Un conjunto de funciones para devolver el contenido o mostrarlo:

```
lee :: IO String
lee = readFile "Ejemplo_1.txt"

main :: IO ()
main = do
    contenido <- lee
    putStrLn contenido
```

  De modo que luego podamos pedir al intérprete:

```
Prelude> lee
"Este fichero tiene tres lineas\nesta es la segunda y\nesta es la tercera."

Prelude> main
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.
```

Escritura en ficheros con `writeFile`

- Podemos escribir una cadena a un fichero con `writeFile`:

```
Prelude> :type writeFile
writeFile :: FilePath -> String -> IO ()

Prelude> let texto = "Hay\ntres lineas\nde texto"

Prelude> writeFile "Ejemplo_2.txt" texto

Prelude> muestraContenidoFichero "Ejemplo_2.txt"
Hay
tres lineas
de texto
```

Ejemplos lectura y escritura en ficheros

Ejemplo: pasar a mayúscula

Definir una función `(aMayuscula x y)` que tome dos ficheros de entrada, lea del primero, pase todos los caracteres a mayúscula, y escriba en el segundo la cadena resultante.

Ejemplos lectura y escritura en ficheros

Ejemplo: pasar a mayúscula

Definir una función `(aMayuscula x y)` que tome dos ficheros de entrada, lea del primero, pase todos los caracteres a mayúscula, y escriba en el segundo la cadena resultante.

Debería funcionar de esta manera:

```
Prelude> muestraContenidoFichero "Ejemplo_1.txt"
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.

Prelude> aMayuscula "Ejemplo_1.txt" "Ejemplo_3.txt"
Prelude> muestraContenidoFichero "Ejemplo_3.txt"
ESTE FICHERO TIENE TRES LINEAS
ESTA ES LA SEGUNDA Y
ESTA LA TERCERA.
```

Ejemplos lectura y escritura en ficheros

Ejemplo: pasar a mayúscula

Definir una función `(aMayuscula x y)` que tome dos ficheros de entrada, lea del primero, pase todos los caracteres a mayúscula, y escriba en el segundo la cadena resultante.

Debería funcionar de esta manera:

```
Prelude> muestraContenidoFichero "Ejemplo_1.txt"
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.

Prelude> aMayuscula "Ejemplo_1.txt" "Ejemplo_3.txt"
Prelude> muestraContenidoFichero "Ejemplo_3.txt"
ESTE FICHERO TIENE TRES LINEAS
ESTA ES LA SEGUNDA Y
ESTA LA TERCERA.
```

¿Cómo podemos hacerlo?

Ejemplos lectura y escritura en ficheros

Ejemplo: pasar a mayúscula

Definir una función `(aMayuscula x y)` que tome dos ficheros de entrada, lea del primero, pase todos los caracteres a mayúscula, y escriba en el segundo la cadena resultante.

Debería funcionar de esta manera:

```
Prelude> muestraContenidoFichero "Ejemplo_1.txt"
Este fichero tiene tres lineas
esta es la segunda y
esta la tercera.

Prelude> aMayuscula "Ejemplo_1.txt" "Ejemplo_3.txt"
Prelude> muestraContenidoFichero "Ejemplo_3.txt"
ESTE FICHERO TIENE TRES LINEAS
ESTA ES LA SEGUNDA Y
ESTA LA TERCERA.
```

¿Cómo podemos hacerlo?

```
import Data.Char (toUpper)

aMayuscula f1 f2 = do
    contenido <- readFile f1
    writeFile f2 (map toUpper contenido)
```

Ejemplos lectura y escritura en ficheros

Ejemplos (II) - Ordenación de líneas

Ejemplo: ordenar líneas del fichero

Definir una función `(ordenaFichero f1 f2)` que lea el contenido de `f1`, ordene sus líneas, y lo guarde como `f2`.

Ejemplos lectura y escritura en ficheros

Ejemplos (II) - Ordenación de líneas

Ejemplo: ordenar líneas del fichero

Definir una función `(ordenaFichero f1 f2)` que lea el contenido de `f1`, ordene sus líneas, y lo guarde como `f2`.

Debería funcionar de esta manera:

```
Prelude> muestraContenidoFichero "Ejemplo_4a.txt"
Juan Ramos
Ana Ruiz
Luis Garcia
Blanca Perez

Prelude> ordenaFichero "Ejemplo_4a.txt" "Ejemplo_4b.txt"
Prelude> muestraContenidoFichero "Ejemplo_4b.txt"
Ana Ruiz
Blanca Perez
Juan Ramos
Luis Garcia
```

Ejemplos lectura y escritura en ficheros

Ejemplos (II) - Ordenación de líneas

Trata de hacerlo sabiendo que existe la función `lines` para pasar una cadena a una lista de cadenas (una por cada línea), y `unlines`, para el proceso contrario.

Ejemplos lectura y escritura en ficheros

Ejemplos (II) - Ordenación de líneas

Trata de hacerlo sabiendo que existe la función `lines` para pasar una cadena a una lista de cadenas (una por cada línea), y `unlines`, para el proceso contrario.

```
Prelude> :type lines
lines :: String -> [String]
Prelude> :type unlines
unlines :: [String] -> String

Prelude> lines "Primera\nSegunda\nTercera"
["Primera","Segunda","Tercera"]

Prelude> unlines ["Primera","Segunda","Tercera"]
"Primera\nSegunda\nTercera\n"
```

Ejemplos lectura y escritura en ficheros

Ejemplos (II) - Ordenación de líneas

```
ordenaFichero :: FilePath -> FilePath -> IO ()
ordenaFichero f1 f2 = do
    contents <- readFile f1
    writeFile f2 (ordenaLineas (lines contents))

ordenaLineas :: [String] -> String
ordenaLineas lineas = unlines (sort lineas)
```

Las funciones `words` y `unwords`

```
Prelude> :type words
words :: String -> [String]
Prelude> :type unwords
unwords :: [String] -> String

Prelude> words "ayer fue   martes"
["ayer","fue","martes"]
Prelude> unwords it
"ayer fue martes"
```

Ejemplo de uso de `words`/`unwords`

El procedimiento (`tablaCuadrados f n`) escribe en el fichero `f` los cuadrados de los `n` primeros números naturales positivos. Por ejemplo:

```
Prelude> tablaCuadrados "cuadrados.txt" 9
Prelude> muestraContenidoFichero "cuadrados.txt"
(1,1) (2,4) (3,9) (4,16) (5,25) (6,36) (7,49) (8,64) (9,81)
```

El programa es:

```
tablaCuadrados :: FilePath -> Int -> IO ()
tablaCuadrados f n =
    writeFile f (listaDeCuadrados n)

listaDeCuadrados :: Int -> String
listaDeCuadrados n =
    unwords (map show [(x,x*x) | x <- [1..n]])
```

Uso de `lines`/`unlines` y de `words`/`unwords`

El procedimiento `(tablaCuadrados2 f n)` escribe en el fichero `f` los cuadrados de los `n` primeros números naturales positivos, uno por línea. Por ejemplo:

```
Prelude> tablaCuadrados2 "cuadrados2.txt" 9
Prelude> muestraContenidoFichero "cuadrados2.txt"
(1,1)
(2,4)
(3,9)
(4,16)
(5,25)
```

El programa es:

```
tablaCuadrados2 :: FilePath -> Int -> IO ()
tablaCuadrados2 f n =
    writeFile f (listaDeCuadrados2 n)

listaDeCuadrados2 :: Int -> String
listaDeCuadrados2 n =
    unlines (map show [(x,x*x) | x <- [1..n]])
```

Un último ejemplo

El procedimiento (`tablaLogaritmos f ns`) escribe en el fichero `f` los cuadrados de los números de la lista `ns`, uno por línea. Por ejemplo:

```
Prelude> tablaLogaritmos "z.txt" [1,3..20]
Prelude> muestraContenidoFichero "z.txt"
+----+----------------+
| n  | log(n)         |
+----+----------------+
|  1 | 0.000000000000 |
|  3 | 1.098612288668 |
|  5 | 1.609437912434 |
|  7 | 1.945910149055 |
|  9 | 2.197224577336 |
| 11 | 2.397895272798 |
| 13 | 2.564949357462 |
| 15 | 2.708050201102 |
| 17 | 2.833213344056 |
| 19 | 2.944438979166 |
+----+----------------+
```

Un último ejemplo

Solución

El programa es:

```
tablaLogaritmos :: FilePath -> [Int] -> IO ()
tablaLogaritmos f ns = do
    writeFile f (tablaLogaritmosAux ns)

tablaLogaritmosAux :: [Int] -> String
tablaLogaritmosAux ns =
    linea
    ++ cabecera
    ++ linea
    ++ concat [printf "| %2d | %.12f |\n" n x
              | n <- ns
              , let x = log (fromIntegral n) :: Double]
    ++ linea

linea, cabecera :: String
linea    = "+----+---------------+\n"
cabecera = "| n  | log(n)        |\n"
```

Bibliografía


H. Daumé III. *Yet Another Haskell Tutorial*, 2006.
Chapter 5: Basic Input/Output


G. Hutton. *Programming in Haskell*. Cambridge University Press, 2007.
Chapter 9: Interactive programs


M. Lipovača. *¡Aprende Haskell por el bien de todos!*.
Apartado 9.2: Ficheros y flujos de datos


B. O'Sullivan, D. Stewart y J. Goerzen. *Real World Haskell*. O'Reilly, 2008.
Chapter 7: I/O


B.C. Ruiz, F. Gutiérrez, P. Guerrero y J.E. Gallardo. Razonando con Haskell. Thompson, 2004..
Capítulo 7: Entrada y salida


S. Thompson. Haskell: The Craft of Functional Programming, Second Edition. Addison-Wesley, 1999.
Chapter 18: Programming with actions