HIT

¿Cansado de romper código que andaba y no saber cómo volver atrás? ¿Harto de recibir mails con archivos adjuntos llamados “TP final posta posta ahora sí 2.0”? ¡No sufra más! Tenemos la solución: HIT es una herramienta Haskell de modificación de archivos y control de versionado que facilita el desarrollo de software.

HIT trabaja con archivos de texto, los cuales están modelados como un Data de dos Strings: su nombre y su contenido.

```
data Archivo = Archivo { nombre :: String, contenido :: String } deriving (Show, Eq)

-- ejemplo
unTpGrupal :: Archivo
unTpGrupal = Archivo "tpGrupal.hs" "listaLarga :: [a] -> Bool \n listaLarga = (>9) . length"
```

Como se observa en el ejemplo, cada renglón del contenido del archivo se separa mediante el caracter de escape **\n**. Esto nos permite utilizar las funciones ya definidas *`words`*, *`unwords`*, *`lines`* y *`unlines`*[1] para trabajar con la información de los archivos.

HIT debe soportar algunas operaciones básicas sobre los archivos:

1. Saber el tamaño de un archivo en bytes, que se computa como la cantidad de caracteres del archivo, multiplicada por 8.
2. Saber si un archivo está vacío.
3. Saber la cantidad de líneas de un archivo.
4. Saber si alguna de las líneas del archivo es *blanca:* esto ocurre cuando está formada sólo por caracteres blancos[2].
5. Saber si un archivo es de extensión *.hs*.

Una vez definidas estas operaciones, la cosa se pone interesante: queremos modelar los *cambios*. Deben poder realizarse las siguientes **modificaciones**:

6. Renombrar un archivo.
7. Agregar una nueva línea al archivo, dados el número de línea donde se insertará y el contenido de la misma.
8. Quitar una línea del archivo, dado el número de la misma.
9. Reemplazar una línea del archivo por otra, dado el número de la misma y el contenido nuevo.
10. Buscar y reemplazar en el archivo, que reemplaza una palabra por otra en todas sus apariciones en el archivo.
11. *Wrappear* las líneas del archivo: si una línea tiene menos de 80 caracteres, queda igual. De lo contrario, se corta esa línea y se agrega abajo una nueva con los caracteres restantes, que también debe ser *wrappeada*.

---

[1]
```
words, lines     ::  String  -> [String]
unwords, unlines :: [String] ->  String
```

[2] Los caracteres blancos son el espacio, el enter, el tab, etc. Tip: investigar la función isSpace Se requiere hacer: import Data.Char

Además, queremos:

12. Saber si una modificación es inútil: esto es cuando al aplicarla sobre un archivo no cambia nada.

Pero HIT no sería un sistema de versionado si no soportara revisiones: una **revisión** es un conjunto ordenado de modificaciones aplicables a un archivo.

13. Aplicar una revisión sobre un archivo: esto nos debería permitir saber cómo queda el archivo después de aplicarle todos los cambios individuales.

Y además HIT permite trabajar sobre **directorios**, los cuales se componen de archivos, y podemos aplicar revisiones sobre un directorio. Una **revisión al directorio** es una serie de revisiones aplicadas a sus archivos, donde a cada archivo se le aplica una revisión distinta. Algunos de los archivos puede que nunca hayan sido modificados, entonces no van a tener revisiones.

Por ejemplo:
si tengo un directorio con un archivo a.txt y otro archivo b.txt, una revisión sobre este directorio puede ser agregar "hola" a a.text en la línea 3 y renombrarlo como a2.txt, y quitarle la línea 2 a b.txt.

14. Determinar el archivo más grande de un directorio al aplicarle una revisión a todos los archivos del directorio.
15. Dada una "*revisión de directorio*", al aplicar las modificaciones correspondientes a esa revisión, indicar cuál es el archivo perteneciente al directorio que tiene más diferencia de tamaño, respecto a como era previo a la revisión.
16. Aplicar una serie de revisiones sobre un directorio. Notar que no es lo mismo que aplicar una modificación, porque puede que en una revisión hayan sido modificados ciertos archivos, y que en otra hayan sido modificados otros.

Se pide:

- Desarrollar el código
- Definir la firma (el tipo) de todas las funciones usadas, creando abstracciones necesarias para incrementar la expresividad.
- Escribir un ejemplo de consulta por cada punto