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.

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

unSimulacro :: Archivo
unSimulacro = Archivo "HIT.hs" "esLarga :: [a] -> Bool\nesLarga = (>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 **estaVacio**.
3. Saber la **cantidadDeLineas** de un archivo.
4. Saber si un archivo **tieneLineasBlancas**. Esto ocurre cuando alguna de sus líneas es *blanca, es decir* que está formada sólo por caracteres blancos[2].
5. Saber si un archivo **esArchivoHaskell**, es decir que su extensión es *.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. **agregarLinea** a un archivo dado el número de línea donde se insertará y el contenido de la misma.
8. **quitarLínea** de un archivo, dado el número de la misma.
9. **reemplazarLinea** de un archivo por otra, dado el número de la misma y el contenido nuevo.
10. **buscarYReemplazar** 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] Se puede usar la función isSpace que se encuentra en Data.Char que nos dice si un caracter es blanco.

Además, queremos:

12. Saber si una modificación **esInutil**: 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. **aplicarRevisión** sobre un archivo: esto nos debería permitir saber cómo queda el archivo después de aplicarle todos los cambios individuales.
14. Teniendo archivos de infinitas líneas como son los siguientes:

```
archivoInfinito1 = Archivo "infinito1.txt" (cycle "soy una linea\n")
archivoInfinito2 = Archivo "infinito2.txt" (cycle "soy\notra linea\n  \n")
```

Indicar cuáles de las siguientes consultas:

- funciona y muestra el resultado por pantalla (indicar cuál sería el resultado);
- funciona y muestra el resultado por pantalla pero nunca termina de mostrarlo;
- funciona y no muestra el resultado por pantalla (se tilda);
- no funciona (tira error).

```
a) tamanio archivoInfinito2
b) estaVacio archivoInfinito1
c) esArchivoHaskell (nombre archivoInfinito2)
d) tieneLineasBlancas archivoInfinito1
e) tieneLineasBlancas archivoInfinito2
f) take 28 . contenido . reemplazarLinea 3 "la linea tres" $ archivoInfinito2
```

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