FUNDAMENTOS DE COMPUTACIÓN
TRABAJO ENTREGABLE
JUNIO 2024

El objetivo de este trabajo es trabajar con listas y árboles. El mismo tiene un puntaje de 9 puntos y debe ser realizado en grupos de hasta dos estudiantes.
La entrega se realizará por Aulas antes del 21 de junio a las 21 hrs.

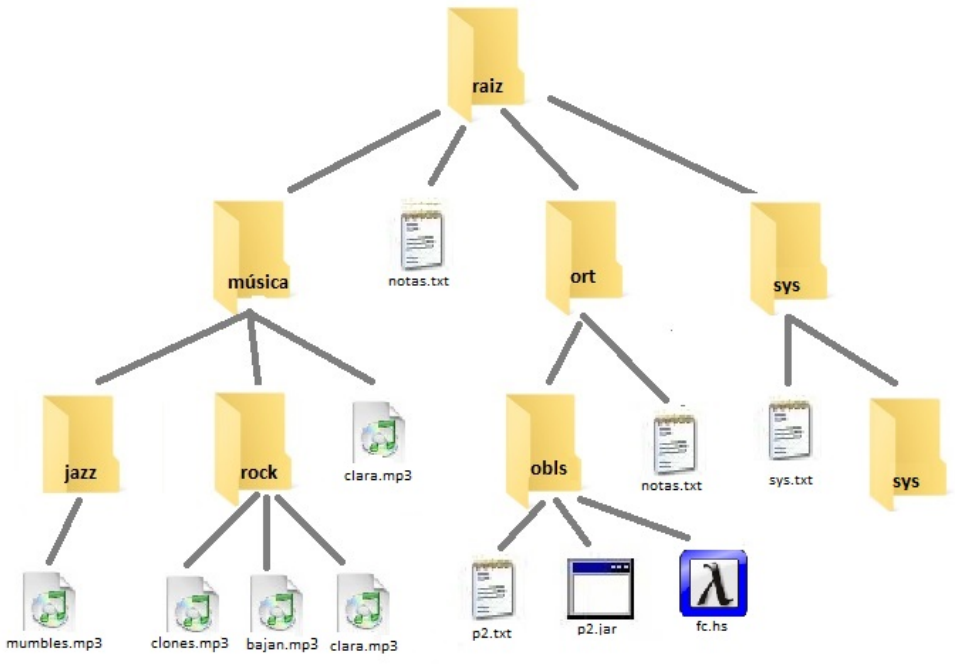
Recomendamos utilizar las funciones del Preludio de Haskell vistas en clase para booleanos y listas, así como definir funciones auxiliares cuando sea necesario, para hacer más legible el código.
Se tendrá en cuenta el estilo de programación y el nivel de abstracción utilizado en la definición de las funciones.

1. Descripción general

Se desea definir funciones que implementen un sistema de archivos (en inglés *File System*).
Los File Systems son estructuras jerárquicas que proporcionan una organización lógica para acceder a la información en un sistema operativo.

Los elementos básicos de un File System son los archivos y las carpetas (o directorios):

- Un **archivo** es una colección de datos con ciertas propiedades: nombre, tipo, contenido, tamaño, etc.
- Una **carpeta** tiene propiedades como nombre, tipo, protección, y su característica principal es que puede contener archivos u otras carpetas.



En la figura anterior mostramos un ejemplo de un File System, representado como un árbol, donde las carpetas se muestran como íconos amarillos, y los distintos tipos de archivos con diferentes íconos representativos de su contenido.

2. File Systems

Un File System se representará como un tipo de árboles en Haskell con dos constructores: uno para archivos y otro para carpetas.
Para simplificar la representación, los archivos tendrán como atributo su nombre y su extensión, y las carpetas poseerán un nombre y una lista con los archivos y carpetas que contienen.

Definimos primero en Haskell el tipo de los nombres y el tipo de las extensiones:

```
type Nombre = String

data Ext where { Txt::Ext ; Mp3::Ext ; Jar::Ext ; Doc::Ext ; Hs::Ext }
  deriving (Eq,Show)
```

Luego, definimos el tipo de los File Systems:

```
data FS where {  A :: (Nombre,Ext) -> FS;
                 C :: Nombre -> [FS] -> FS }
  deriving (Eq,Show)
```

De esta forma, Ejemplo,

- el archivo fc.hs se representará como: `A ("fc",Hs)`
- el carpeta sys se representará como: `C "sys" [ A("sys",Txt), C "sys" [] ]`

Notar que puede haber nombres repetidos, tanto de archivos como de carpetas, siempre que no se encuentren en la misma carpeta, y en el caso de los archivos, que sean de distinto tipo.

El resto de los componentes del File System del ejemplo puede ser definido del siguiente modo:

```
cjazz::FS
cjazz = C "jazz" [ A("mumbles",Mp3) ]

crock::FS
crock = C "rock" [ A("clones",Mp3), A("bajan",Mp3), A("clara",Mp3) ]

cmusica::FS
cmusica = C "musica"[ cjazz, crock, A("clara",Mp3) ]

cort::FS
cort = ...
.
.
craiz::FS
craiz = ...
```

Se pide:

1. Completar la definición del File System del ejemplo, definiendo sus componentes.

2. Definir la función `nombre :: FS -> Nombre`, que dado un File System, devuelve el nombre de su raiz, en caso de tratarse de una carpeta, o el nombre completo (incluyendo su extensión, en minúscula) en caso de tratarse de un archivo.
   Ejemplo:
   ```
   nombre cmusica = "musica"
   nombre (A ("clara",Mp3)) = "clara.mp3"
   ```

3. Definir la función `contenido :: FS -> [Nombre]`, que recibe una carpeta y devuelve una lista con los nombres de los archivos y carpetas que contiene. Si el argumento no es una carpeta se debe devolver un mensaje de error.
   Ejemplo:
   ```
   contenido craiz = ["musica","notas.txt","ort","sys"]
   contenido (A("clara",Mp3)) = error "no es una carpeta"
   ```

4. Definir la función `cantA :: FS -> Int`, que dado un File System, retorna la cantidad total de archivos que contiene (incluyendo los que estén en subcarpetas).
   Ejemplo:
   ```
   cantA cmusica = 5
   cantA craiz = 11
   ```

5. Definir la función `pertenece :: Nombre -> FS -> Bool`, que dado un Nombre y un File System, retorna `True` si existe algún archivo (sin importar la extensión) o carpeta con ese nombre en el File System.
   Ejemplo:
   ```
   pertenece "clones" cmusica = True
   pertenece "sys" cmusica = False
   ```

6. Definir la función `valido :: FS -> Bool`, que recibe un File System y verifica que no haya nombres repetidos de archivos o carpetas en ninguna carpeta del mismo. Esto quiere decir que en ninguna carpeta puede haber dos archivos con el mismo nombre y extensión, o dos carpetas con el mismo nombre.
   Ejemplo:
   ```
   valido craiz = True
   valido (C "X" [C "ort" [], cort, A("a",Hs)]) = False
   valido (C "X" [A("b",Hs),C "Y" [A("a",Hs),A("a",Jar),A("a",Hs)]]) = False
   ```

7. `archivosExt :: Ext -> FS -> [Nombre]`, que dado un File System y una extensión, retorna una lista con los nombres de todos los archivos que tienen esa extensión en el File System (con repeticiones, en caso de haberlas).
   Ejemplo: `archivosExt Txt craiz = ["notas","p2","notas","sys"]`, en este u otro orden.

8. `cambiarNom :: Nombre -> Nombre -> FS -> FS`, que dados un File System, un nombre y un nuevo nombre, cambia el nombre de todos los archivos y carpetas del File System que tienen el primer nombre por el segundo nombre, manteniendo la extensión en el caso de los archivos.
   Ejemplo: `cambiarNom "sys" "s" csys = C "s" [A ("s",Txt),C "s" []]`

9. `nivelesC :: FS -> Int`, que dado un File System, retorna la cantidad máxima de carpetas por los que puede pasar desde la raiz de la carpeta recibida hasta terminar el árbol (incluyendo a la raiz). En caso que no haya carpetas por tratarse de un solo archivo, se debe devolver 0.

Ejemplo:
```
nivelesC craiz = 3
nivelesC (C "sys" []) = 1
nivelesC (A("fc",Hs)) = 0
```

10. `borrar :: Nombre -> FS -> FS`, que dados un File System y un nombre, borra todos archivos o carpetas *debajo* de la raiz de la carpeta recibido que tengan ese nombre (incluyendo el contenido de las carpeta, cuando corresponda). En caso de que el File System no sea una carpeta, se deberá devolver el mismo File System recibido.
Ejemplo:
```
borrar "obls" cort = C "ort" [A("notas",Txt)].
borrar "jazz" cjazz = C "jazz" [A("mumbles",Mp3)]
borrar "sys" csys = C "sys" [A ("sys",Txt)]
borrar "sys.txt" csys = C "sys" [C "sys" []]
borrar "p2.txt" cort =
          C "ort" [C "obls" [A("p2",Jar),A("fc",Hs)],A("notas",Txt)]
```

11. `ordenar :: FS -> FS`, que dado un File System, lo devuelve ordenado alfabéticamente por nombre en todos sus niveles. Para ordenar la lista, recomendamos adaptar alguno de los métodos de ordenamiento visto en el práctico de Sorting.
Ejemplo:
```
ordenar cort = C "ort" [A("notas",Txt),C "obls" [A("fc",Hs),A("p2",Jar),A("p2",Txt)]]
ordenar craiz = C "raiz" [C "musica" [A ("clara",Mp3),
                                      C "jazz" [A ("mumbles",Mp3)],
                                      C "rock" [A ("bajan",Mp3),A ("clara",Mp3),A ("clones",Mp3)]],
                          A ("notas",Txt),
                          C "ort" [A ("notas",Txt),
                                   C "obls" [A ("fc",Hs),A ("p2",Jar),A ("p2",Txt)]],
                          C "sys" [C "sys" [],A ("sys",Txt)]]
```

3. Entregables

- El trabajo deberá realizarse en grupos de hasta dos estudiantes.

- Se pueden utilizar las funciones del Preludio de Haskell definidas para booleanos y listas. Cualquier otra función auxiliar que se necesite utilizar aparte de las de Preludio debe ser definida y se debe explicar qué hace (en forma de comentario en el código).

- La entrega deberá realizarse por Aulas antes del **21/6/24 a las 21:00 hs.**

- Deberá subirse a Aulas **un único archivo Haskell por equipo** con el código fuente de la solución. El archivo debe incluir los nombres y números de estudiantes como comentarios al principio del mismo.

- En Aulas se encuentra el archivo `FS.hs` con las funciones que deben implementarse. Para facilitar la corrección deberá usarse el mismo como template.

- **No se corregirán archivos que no compilen, por lo que recomendamos comentar el código que no compile y dejar como `undefined` las funciones no implementadas.**

- Para comprobar autoría, este entregable tendrá una instancia de defensa durante el parcial.

4. Anexo: algunas funciones útiles del Preludio de Haskell

```
elem :: Eq a => a -> [a] -> Bool

(++) :: [a] -> [a] -> [a]

map :: (a->b) -> [a] -> [b]

filter :: (a->Bool) -> [a] -> [a]

all :: (a -> Bool) -> [a] -> Bool

any :: (a -> Bool) -> [a] -> Bool

and :: [Bool] -> Bool

or :: [Bool] -> Bool

sum :: [Int] -> Int

concat :: [[a]] -> [a]

maximum :: Ord a => [a] -> a
```