Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información
CI3661 - Taller de Lenguajes de Programación I
Enero-Marzo 2014

Programación Funcional - Entrega 2
Pixel Displays

En esta etapa del proyecto, Ud. va a construir un programa en Haskell que realizará el despliegue del *led display* a través de una ventana gráfica. Su programa recibira argumentos en la línea de comandos: el primero indicará el nombre de un archivo que contiene la especificación del *font* a utilizar, y el resto serán uno o más nombres de archivos que indican mensajes de texto combinados con la secuencia de efectos especiales a aplicar sobre el mismo. El programa ha de correr desde la línea de comandos, por lo que necesitará ser compilado empleando GHC.

Desarrollo de la Implementación

Modificaremos el tipo `Pixels` para convertirlo en algebráico, como los elementos básicos para construir nuestros *led displays*, i.e.

```
data Pixels = Pixels { color :: Color, dots :: [[Pixel]] }
```

siendo `Pixel` un nuevo tipo algebráico definido como

```
data Pixel = Pixel { on :: Boolean }
```

Esto implica que deberá modificar las firmas y posiblemente la definición de todas las funciones provistas en la primera entrega para reflejar el cambio en el tipo de datos. El tipo auxiliar `Color` para representar el color es el mismo de la librería HGL [1].

¿Cómo leer la descripción de un *font*?

En la primera entrega, la definición del *font* se tomaba a partir del valor `fontBitmap` provisto. En esta entrega ya no se hará uso de ese valor, sino que será necesario escribir una función `readFont` que permita obtener la representación en pixels de los caracteres particulares del alfabeto a partir de un archivo, construyendo un `Data.Map` que los contenga para futura referencia.

```
readFont :: Handle -> IO (Map Char Pixels)
```

El archivo con la definición del *font* tiene un formato muy simple:

- La primera línea indica las dimensiones de cada *font* usando dos números, el primero indica el número de columnas y el segundo el número de filas. Puede haber espacios en blanco antes y después de los números, los cuales deben ser ignorados, i.e.

  ```
  5 7
  ```

Antes del 5 y después del 7 puede haber *cualquier* cantidad de espacios en blanco o tabuladores. Entre el 5 y el 7 puede haber uno o más espacios en blanco o tabuladores. El programa debe verificar que en efecto hay dos números positivos antes de proseguir; en caso contrario, el programa debe abortar indicando el problema.

- Después de la línea inicial, sigue una cantidad arbitraria de líneas que definen cada uno de los símbolos del *font*. Para cada símbolo del *font*, encontrará una línea indicando el caracter a representar, entre comillas dobles, seguido de tantas líneas como filas tenga la representación, cada una de las cuales con tantas columnas como se haya indicado. Se utilizarán espacios en blanco y asteriscos para indicar pixels apagados o encendidos, respectivamente. Así, para representar la X podría tenerse,

```
"X"
*   *
*   *
 * *
  *
 * *
*   *
*   *
```

  El programa debe verificar que hay exactamente un símbolo entre comillas, y que hay tantas líneas, con tantas columnas, según la especificación de tamaño al comienzo del archivo; en caso contrario, el programa debe abortar indicando el problema.

- El programa ha de procesar las líneas de asteriscos y espacios en blancos para convertirla en un valor `Pixels`, el cual será almacenado en un `Data.Map` usando al símbolo (en nuestro caso `X,` sin las comillas) como la clave. El programa debe iniciar con un `Data.Map` vacío, el cual se irá llenando a medida que se procesa el archivo de definición del *font.*

- El color por omisión durante la carga del *font*, es el blanco.

- Los archivos de definición de *font* no necesariamente tienen que tener definiciones para todos los símbolos. Esto es, podría contener sólo letras, sólo números, o cualquier subconjunto de las letras y los números. Su programa no debe hacer ninguna suposición en cuanto a cuántos símbolos están definidos en el archivo, más allá de que debe haber al menos un símbolo definido. Si el archivo contiene más de una definición para el mismo símbolo, debe prevalecer la *última* definición encontrada.

Esto implica que será necesario modificar la función `font` para que obtenga la información a partir de un `Data.Map`; i.e.

```
font :: Map Char Pixels -> Char -> Pixels
```

Note que la función `font` *siempre* debe producir un resultado del tipo `Pixels`, de manera que si se solicita el *font* para un caracter que no está definido, debe retornarse un `Pixels` que tenga todos los pixels encendidos.

¿Cuáles efectos especiales?

Definiremos un tipo de datos recursivo para representar los efectos especiales a aplicar, de la forma

```
data Effects = Say String
             | Up
             | Down
             | Left
             | Right
             | Backwards
             | UpsideDown
             | Negative
             | Delay Integer
             | Color Color
             | Repeat Integer [Effects]
             | Forever [Effects]
```

permitiendo:

- Indicar el mensaje a mostrar (`Say`).
- Representar efectos individuales sobre el mensaje actualmente mostrado (`Up`, `Down`, `Left`, `Right`, `Backwards`, `UpsideDown`, `Negative`).
- Hacer una pausa en milisegundos (`Delay`).
- Repetir una lista de eventos un número específico de veces (`Repeat`) o bien infinitamente (`Forever`).

La intención de este tipo de datos es permitir la escritura de instrucciones de presentación, e.g.

```
Forever [ Say "Hello world!",
          Repeat 4 [ Delay 500, Color Yellow, Up,
                     Delay 500, Color Blue, Right,
                     Delay 500, Color Red, Down,
                     Delay 500, Color White, Left
                   ],
          Say "Hola mundo!",
          Negative ]
```

La especificación del mensaje a mostrar en el *led display* y los efectos a aplicar, serán suministrados a través de uno o más archivos de texto. Por lo tanto, será necesario implantar una función `readDisplayInfo` capaz de obtener la información necesaria a partir de un archivo.

```
readDisplayInfo :: Handle -> IO [Effects]
```

El archivo en cuestión puede incluir cero o más valores del tipo `Effects`, los cuales deben ser leídos y retornados como una lista en el mismo orden en que aparecen en el archivo.

¿Qué hacer con el *font* y efectos especiales?

Para este proyecto, la salida del *led display* será representada en una ventana gráfica[1]. Para ello, aproveche la librería HGL para construir una ventana del tamaño adecuado para contener el mensaje más largo que deba aparecer en el *led display* y presentarlo allí. Cada `Pixel` debe ocupar un espacio de tres pixels reales de ancho por tres pixels reales de alto, con un margen de un pixel real en cada dirección. La función principal que se encargue de ese trabajo debe tener la firma

```
ledDisplay :: Map Char Pixels -> [Effects] -> IO ()
```

La función `ledDisplay` debe recibir el mapa con la definición del *font* así como la lista de efectos resultado de combinar todos los archivos de efectos suministrados como argumentos de línea de comandos. Si la cantidad de efectos a aplicar es finita, el programa debe esperar a que el usuario oprima cualquier tecla antes de cerrar la ventana y terminar. Independientemente de la cantidad de efectos a aplicar, el usuario debe poder salir del programa de manera *normal* oprimiendo la tecla Escape.

Sugerencias

- Aproveche las librerías `Data.Map` y `Graphics.HGL` usando nombres completos (`import qualified`).
- Tal como se discutió en clase haga un esfuerzo por separar de manera clara el procesamiento puro de `Pixels`, la creación de secuencias de eventos gráficos a partir de los tipos de datos, y la ejecución de las acciones gráficas en `IO`. No solamente resultará en un estilo más claro de programación, sino que le facilitará el desarrollo incremental de su proyecto.
- Va a necesitar instalar la librería `Graphics.HGL` usando `cabal`. Hágalo ya.

Entrega de la Implementación

Ud. debe entregar un archivo `.tar.gz` o `.tar.bz2` (**no** puede ser `.rar`, ni `.zip`) que al expandirse genere un directorio con el nombre de su grupo (e.g. `G42`) dentro del cual encontrar **solamente**:

- El archivo `Pixels.hs` conteniendo el código fuente Haskell para implantar los tipos de datos abstractos `Pixel` y `Pixels`, además de las funciones que operan sobre ellos.
- El archivo `Effects.hs` conteniendo el código fuente Haskell para implantar el tipo de datos `Effects`, además de las funciones que operan sobre ellos. El módulo **no** puede manipular la estructura interna de `Pixel` o `Pixels` directamente.
- El archivo `led-display.hs` conteniendo el código fuente Haskell para el programa principal. Todas las operaciones de I/O deben estar contenidas en este archivo, con el necesario manejo de excepciones.
- Se evaluará el correcto estilo de programación:

[1] Note que las funciones `pixelToString` y `pixelListToString` ya no serán necesarias

- Indentación adecuada y consistente en *cualquier* editor ("profe, en el mío se ve bien" es inaceptable).
- Los módulos solamente deben exportar los tipos de datos y funciones definidas en este enunciado. Cualquier función auxiliar (que **seguro** le harán falta) deben aparecer bien sea en el contexto local de un `where` o como una función privada al módulo.
- Todas las funciones deben tener su firma, con el tipo más general posible.
- Todas las funciones deben escribirse aprovechando constructores de orden superior (`map`, `fold`, `filter`, `zip`, secciones y composición de funciones) evitando el uso de recursión directa.

- Los archivos **deben** estar completa y correctamente documentado utilizando la herramienta Haddock. Ud. **no** entregará los documentos HTML generados, sino que deben poder **generarse** de manera automática incluyendo acentos y símbolos especiales. Es **inaceptable** que la documentación tenga errores ortográficos.
- El programa será compilado utilizando `ghc --make` o el Makefile suministrado por Ud. hasta construir el programa ejecutable final, el cual será invocado desde la línea de comandos

  ```
  $ led-display font.txt efectos1.txt efectos2.txt ...
  ```

  El primer archivo indicado en la línea de comandos será utilizado para obtener la definición del *font* a utilizar, mientras que el resto de los archivos (al menos uno) serán utilizados para obtener los efectos especiales a aplicar sobre el mismo.

  Si alguno de los archivos no puede ser leído por la razón que sea, el programa debe indicarlo con un mensaje de error apropiado y finalizar la ejecución. Es **inaceptable** que el programa aborte sin control.
- **Fecha de Entrega.** Viernes 2014-05-09 hasta las 23:59 VET
- **Valor de Evaluación.** Veinticinco (25) puntos.

Referencias

[1] HGL Graphics Library
`https://hackage.haskell.org/package/HGL`