Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información
CI3661 - Taller de Lenguajes de Programación I
Septiembre-Diciembre 2009

Programación Funcional - Entrega 2
Pixel Displays

En esta etapa del proyecto, Ud. va a construir un programa en Haskell que recibe la especificación del *font* a utilizar, el texto y la secuencia de efectos especiales a aplicar sobre el mismo, y realizará el despliegue del *led display* a través de una ventana gráfica. El programa ha de correr desde la línea de comandos, por lo que necesitará ser compilado empleando GHC, bien sea con la ayuda de `hmake` o mediante la construcción de un Makefile a la medida.

Desarrollo de la Implementación

Modificaremos el tipo `Pixels` para basarlo en una lista de `Pixel`, como los elementos básicos para construir nuestros *led displays*, i.e.

```
type Pixels = [[Pixel]]
```

siendo Pixel un nuevo tipo algebráico definido como

```
data Pixel = Empty | Pixel { on :: Boolean, color :: Color }
```

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 para representar el color es el mismo de la librería HGL [1].

Hará falta 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. Así mismo, será necesario modificar la función `font` para que obtenga la información a partir de un `Data.Map`; i.e.

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

- Para cada símbolo del *font*, vendrá una línea que indica 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. No habrá errores de forma en el archivo, esto es, puede asumirse que la cantidad de filas y columnas es concordante con lo indicado en la primera línea. Así, para representar la X podría tenerse,

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

- Al cargar el *font*, se asumirá que debe ser mostrado en color blanco.

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

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

permitiendo representar efectos individuales, una pausa en milisegundos, y una lista de eventos a repetir un número específico de veces o bien infinitamente. El efecto `Invert` corresponde a enceder todo `Pixel` apagado y apagar todo `Pixel` encendido, es decir, representarlo en "video inverso", preservando los colores en uso. Note que además de modificar las funciones existentes para los eventos, será necesario implantar funciones adicionales para los nuevos efectos individuales (inversión y aplicar el cambio de color).

La especificación del mensaje a mostrar en el *led display* y los efectos a aplicar, serán suministrados a través de un archivo 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 (String,Effects)
```

El archivo en cuestión tiene un formato muy simple:

- La primera línea indica el mensaje de texto a mostrar en el led display.

- El resto de las líneas indican los efectos especiales a aplicar sobre el texto.
- Cualquier desviación sobre el formato debe ser reportada como un error fatal.

Para este proyecto, la salida del *led display* será representada en una ventana gráfica[1]. Para ello, debe aprovechar la librería HGL para construir una ventana con las dimensiones necesarias para contener el mensaje del *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

```
displayMessage :: String-> Map Char Pixels-> Effects-> IO ()
```

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:
  - Indentación adecuada y consistente en *cualquier* editor ("profe, en el mío se ve bien" es inaceptable).
  - Los módulos solamente debe 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.

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

- El programa será compilado utilizando `hmake` 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 info.txt
  ```

  El primer archivo indicado en la línea de comandos será utilizado para obtener la definición del *font* a utilizar, mientras que el segundo archivo será utilizado para obtener el mensaje a mostrar y los efectos especiales a aplicar sobre el mismo. El programa se ejecutará hasta que todos los efectos especiales hayan sido aplicados sobre el texto, o bien el usuario presione la tecla Esc, lo que ocurra primero.

  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.** Jueves 2009-11-05 (Semana 7) hasta las 15:29 VET
- **Valor de Evaluación.** Veinticinco (25) puntos.

Referencias

[1] HGL Graphics Library
`http://haskell.cs.yale.edu/graphics/`
`aptitude install ghc6-hgl-dev ghc6-hgl-doc`