LABORATORIO DE LENGUAJES DE PROGRAMACIÓN
UNIVERSIDAD SIMÓN BOLÍVAR

PROYECTO 1 DE HASKELL

De pixeles a LEDs
(20 puntos)

Los pixelex están por todos lados. Están construidos con hileras de LEDs (Light Emitting Diodes), y presentan mensajes de texto o imágenes con rotaciones, desplazamientos, etc.

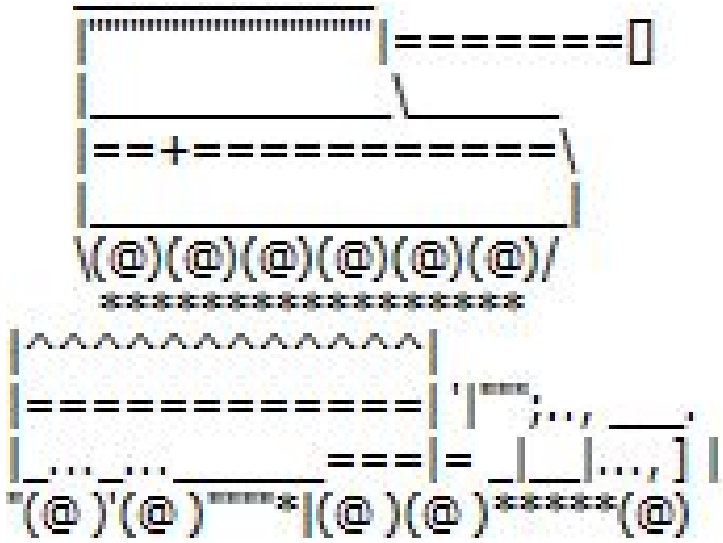

Fig. 1 Representación de una figura en ascii

Ud. va a construir un programa en Haskell que recibe una especificación básica de la forma en la que el texto debería estar representado en el led display y además su color, el resultado debe estar mostrado por pantalla en un String. De esta forma, las operaciones serán realizadas a través del interpretador GHCi.

Implementación

El tipo de datos que se sugiere para Pixeles:

El tipo de datos por definir es el de Pixeles basado en dos listas una de String, como los que van a mostrar por pantalla y otra para el color.

```
data Pixel = Pixel {mensaje:: String,
                    color::String}
```

Sobre este tipo se construiran funciones para la construcción de mensajes.

Tipografía

Es necesario representar todos los caracteres imprimibles de la tabla ASCII, ésto incluye espacios en blanco, de forma que se puedan escribir mensajes arbitarios. Hará falta una función para ésto, *font,* la cual permitirá obtener la representación en pixeles de un caracter particular del alfabeto.

```
 > font 'A' ‘rojo’’
[ "  ***  ",
  "*   *",
  "*   *",
  "*****",
  "*   *",
  "*   *",
  "*   *" ]
```

Por simplicidad sólo hay cuatro colores posibles, rojo negro, amarillo y verde.
El asterisco representa un pixel encendido y el espacio en blanco representa un pixel apagado.
La representacion es una lista que siempre tiene siete (7) elementos, cada uno de los cuales es un String que siempre tiene cinco (5) caracteres. Hemos provisto un archivo Pixels.hs sobre el cual Ud. debe trabajar. En ese archivo encontrara la denicion del valor fontBitmap , que contiene un mapa de bits apropiado; Ud. no puede modicar esa denicion, pero debe aprovecharla para que su funcion font calcule los pixels adecuados.2
Si se pide calcular el font para un caracter fuera del rango imprimible en la tabla ASCII, debe producirse un bloque con todos los pixels encendidos.

Más sobre la implementación

Las salidas deben ser mostradas por la pantalla del interpretador. Por lo que necesitamos convertir nuestro pixel en string en algún punto. Entonces, debes proveer tres funciones:

1. pixelsToString: convierte un valor del tipo Pixel en un string, incluyendo los saltos, por ejemplo:
   > pixelToString (font 'A' ‘rojo’)

"*** \n* *\n* *\n*****\n* *\n* *\n* *"

2. La función pixelListToPixels convierte una lista de Pixeles en un único valor Pixels que lo represente. Combina los elementos individuales de la lista original con una cadena vacía entre ambas

```
 > pixelListToPixels [font 'A' ‘rojo’, font 'B' ‘verde’]
[ " *** ",
  "*   *",
  "*   *",
  "*****",
  "*   *",
  "*   *"
  "*    *",
  ""
  ,
  "**** ",
  "*   *",
  "*   *",
  "**** ",
  "*   *",
  "*   *",
  "**** "]
```

3. La función pixelListToString es muy parecida. Procesa la lista de Pixels, convirtiendo cada elemento a String y luego mezcla todos los resultados incluyendo saltos de linea.

Construir valores complejos

Para establecer un conjunto de combinadores y construir valores complejos (concatenaciones de valores ASCII),

1. Función *concatPixels* recibe una lista de Pixeles y produce un nuevo valor que lo representa (de tipo Pixel también), pero realiza la concatenación horizontall. Por ejemplo:

```
> concatPixels [font 'A', font 'B']
[       " *** **** ",
        "*   **   *",
        "*   **   *",
        "********* ",
        "*   **   *",
        "*   **   *",
        "*   ***** " ]
```

Sobre la entrega

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 codigo fuente Haskell para implantar el tipos de datos Pixels y las funciones que operan sobre el.
Se evaluara el correcto estilo de programacion:

- Indentacion adecuada y consistente en cualquier editor ("profe, en el mo se ve bien" es inaceptable).
- El modulo solamente debe exportar los tipos de datos y funciones denidas en este enunciado. Cualquier funcion auxiliar (que seguro le haran falta) deben aparecer bien sea en el contexto local de un where o como una funcion privada al modulo.
- Todas las funciones deben tener su firma, con el tipo mas general posible.
- Todas las funciones deben escribirse aprovechando constructores de orden superior (map , fold , filter , zip , secciones y composicion de funciones) evitando el uso de recursion directa.

El archivo debe estar completa y correctamente documentado utilizando la herramienta Haddock. Ud. no entregara los documentos HTML generados, sino que deben poder generarse de manera automatica incluyendo acentos y smbolos especiales. La documentación no debe contener errores ortográficos!.

Fecha de entrega: Lunes de semana 7