De Pixel a Led

Fernando Lovera
Sergio Terán

Enero - Marzo 2018

1. Generalidades

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.
En la Figura 1 pueden ver una representación gráfica de pixeles (tomada de pixels arts). Esta figura sirve para ilustrar cómo se hace el display de pixels en GNU.

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, el resultado debe estar mostrado por pantalla en un String. De esta forma, las operaciones serán realizadas a través del interpretador GHCi.

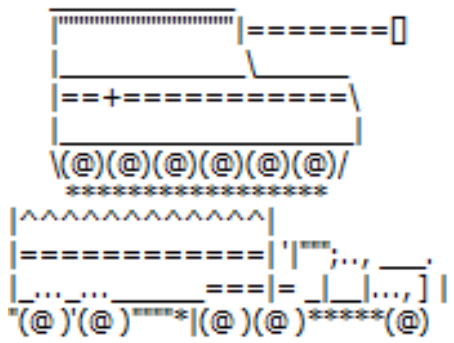

Figura 1: Representación de una figura en ascii

2. Implementación

El tipo de datos que se sugiere para Pixeles:
El tipo de datos por definir es el de Pixeles basado en una única lista de String. Ésto puede ser un tipo de dato creado por ud. o puede ser simplemente un type (pregunta retórica: cuál es la diferencia entre data y type?)

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

Lo que también puede ser expresado como:

type Pixels = [String]

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

3. Tipografia

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. Esta función se va a llamar font, la cual permitirá obtener la representación en pixeles de un caracter particular del alfabeto. La indentación del resultado es para comprender la utilidad de la función. Aparecerían todos los elementos de la lista uno detrás del otro.
prelude>font 'A'

```
[ " *** ",
  "*   *",
  "*   *",
  "*****",
  "*   *",
  "*   *",
  "*   *" ]
```

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.

4. 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:

- **pixelsToString**: convierte un valor del tipo Pixel en un string, incluyendo los saltos, por ejemplo:

  >pixelToString (font 'A')
  "*** * * * * ***** * * * * * * "

- 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', font 'B']
  [ " *** ",
    "*   *",
    "*   *",
    "*****",
    "*   *",
    "*   *"
    "*   *",
    "",
    "**** ",
    "*   *",
    "*   *",
    "**** ",
    "*   *",
    "*   *",
    "**** "]
```

- 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.

5. Construir valores complejos

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

- Función concatPixels recibe una lista de Pixeles y produce un nuevo valor que lo representa (de tipo Pixel también), pero realiza la concatenación horizontall.[Ejemplo en la página siguiente]

Por ejemplo:
>concatPixels [font 'A', font 'B']

```
[ " *** **** ",
  "*   **   *",
  "*   **   *",
  "********* ",
  "*   **   *",
  "*   **   *",
  "*   ***** " ]
```

- La función messageToPixels convierte una cadena de caracteres en un valor Pixels, `agregando un espacio` en blanco entre caracteres. Por ejemplo:
  >messageToPixels "AB"
  Pondría un espacio entre los pixeles A y B.

```
[" ***   **** ",
 "*   * *    *",
 "*   * *    *",
 "***** **** ",
 "*   * *    *",
 "*   * *    *",
 "*   **** " ]
```

6. Efectos especiales

Nuestro led display estar ıa incompleto sin algunos efectos especiales sobre los mensajes. En este sentido, nos interesa disponer de siete transformaciones:

- **Up**: Up La función up que desplaza una hilera hacia arriba.
- **Down**:La función down que desplaza una hilera hacia abajo.
- **Left**: La función left desplaza la pri comeralumna hacia la derecha. La primera columna pasa a ser la última.

  m **Right**: La función right desplaza la última columna hacia la izquierda. La última columna pasa a ser la primera.
- **upsideDown** La función upsideDown invierte el orden de las filas.

- **backwards** invierte el orden de las columnas.
- **negative** intercambia blancos por asteriscos y viceversa.

7. 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

- Indentación.
- Uso de módulos y funciones auxiliares.
- Correcta expresión de firmas en las funciones.
- Uso de funciones de orden superior (map, fold, etc.), lo que quiero decir, es evitar el uso de recursión directa-eso estaba bien en la tarea.
- Finalmente, pero no menos importante, documentación agregada al proyecto.

La fecha de entrega se desplaza dos días, y queda para el Miércoles de semana 7.