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

Programación Funcional - Entrega 1
Pixel Displays

Los *Pixel Displays* están por todos lados (autobuses, aeropuertos, centros comerciales, etc.). Están construidos con hileras de LEDs (*Light Emitting Diodes*), y presentan sus mensajes de texto o imágenes con diversas rotaciones, desplazamientos, etc.

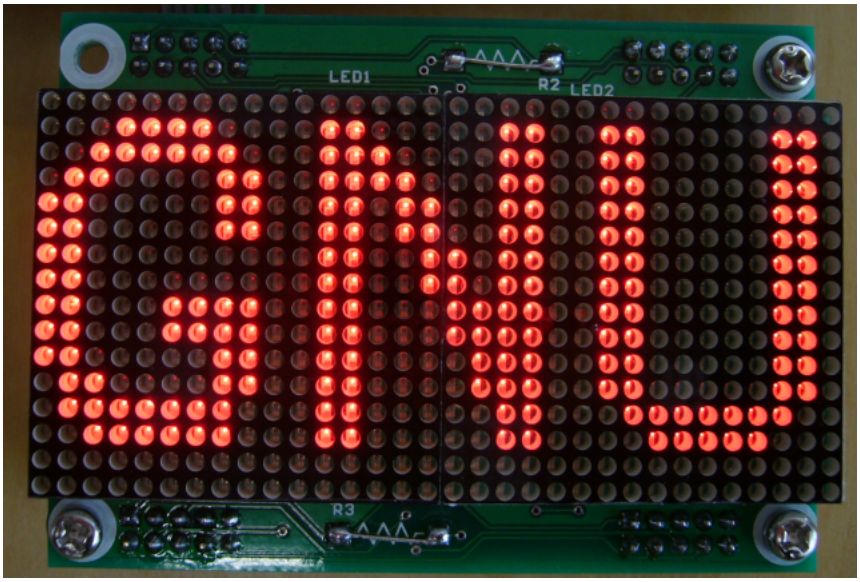

Figura 1: The GNU Pixel display

En esta etapa del proyecto, Ud. va a construir un programa en Haskell que recibe la especificación básica de la forma en la cual el texto debe ser presentado en el *led display*, mostrando en pantalla el resultado como una secuencia de caracteres. Esto permitirá que todas las operaciones sean realizadas a través del interpretador GHCi.

Desarrollo de la Implementación

Un tipo de datos para Pixels

Comenzaremos por definir un tipo `Pixels` basado en una lista de `String`, como los elementos básicos para construir nuestros *led displays*, i.e.

```
type Pixels = [String]
```

y sobre ese tipo será necesario definir algunas funciones fundamentales para la construcción de los mensajes a desplegar.

Tipografía desde mapa de bits

Para nuestro proyecto, es necesario poder representar todos los caracteres imprimibles de la tabla ASCII, incluyendo el espacio en blanco, de manera que podamos escribir mensajes arbitrarios en nuestros *pixel displays*. Hará falta una función `font` que permita obtener la representación en pixels de un caracter particular del alfabeto; i.e.[1]

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

donde el asterisco representa un pixel encendido y el espacio en blanco representa un pixel apagado.

Note que la representación 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 encontrará la definición del valor `fontBitmap`, que contiene un mapa de bits apropiado; Ud. no puede modificar esa definición, pero **debe** aprovecharla para que su función `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.

Mostrando pixels en pantalla

Queremos que la salida del *led display* sea hecha en la pantalla del interpretador. Eso quiere decir que será necesario convertir `Pixels` en `String` para poder desplegarlos. Para esto Ud. debe proveer tres funciones:

- La función `pixelsToString` convierte un valor del tipo `Pixels` en un `String`, combinando los elementos individuales del `Pixel` con retornos del carro en medio. Continuando con nuestro ejemplo

[1] La indentación del resultado es para que se comprenda la utilidad de la función. En realidad aparecerían todos los elementos de la lista uno detrás del otro.

[2] La librería `Data.Bits` puede resultarle útil.

```
> pixelToString (font 'A')
"*** \n*   *\n*   *\n*****\n*   *\n*   *\n*   *"
```

- La función `pixelListToPixels` convierte una lista de `Pixels` en **un** valor `Pixels` que lo represente, combinando los elementos individuales de la lista original con una cadena vacía entre ambos. Continuando con nuestro ejemplo

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

- La función `pixelListToString` es similar. Procesa una lista de `Pixels`, convierte cada elemento a `String` y luego combina todos los resultados incorporando retornos de carro en medio.

Combinadores

Siguiendo con la filosofía de programación funcional de establecer un conjunto de combinadores para construir valores complejos a partir de valores simples, será necesario implantar un par de funciones que permitan combinar de la siguiente forma

- La función `concatPixels` recibe una lista de `Pixels` y produce un nuevo valor `Pixels` que lo representa, pero realizando la concatenación **horizontal**. Esto es, siguiendo con nuestro 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. Siguiendo con nuestro ejemplo

```
> messageToPixels ""
[]
> messageToPixels "A"
[" *** ","*   *","*   *","*****","*   *","*   *","*   *"]
> messageToPixels "AB"[3]
[" ***  **** ",
 "*   * *   *",
 "*   * *   *",
 "***** **** ",
 "*   * *   *",
 "*   * *   *",
 "*   * **** " ]
```

Efectos especiales

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

- La función `up` que desplaza una hilera hacia arriba. Siguiendo con nuestro ejemplo, note como la primera fila pasó a ser la última en

  ```
  > up (font 'A')
  [ "*   *",
    "*   *",
    "*****",
    "*   *",
    "*   *",
    "*   *",
    " *** " ]
  ```

- La función `down` que desplaza una hilera hacia abajo.

- La función `left` que desplaza una columna hacia la izquierda. Siguiendo con nuestro ejemplo, note como la primera columna pasó a ser la última en

  ```
  > left (font 'A')
  [ "*** ",
    "   **",
    "   **",
    "*****",
    "   **",
    "   **",
    "   **" ]
  ```

- La función `right` que desplaza una columna hacia la derecha.

[3] La indentación es para que se vea el espacio en blanco agregado entre las letras. Al ejecutarlo, en realidad se verían los elementos de la lista uno después del otro. Intente

```
putStr $ pixelsToString $ messageToPixels "AB"
```

- La función `upsideDown` que invierte el orden de las filas.
- La función `backwards` que invierte el orden de las columnas.
- La función `negative` que intercambia blancos por asteriscos y viceversa.

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 el tipos de datos `Pixels` y las funciones que operan sobre el.
- Se evaluará el correcto estilo de programación:
  - Indentación adecuada y consistente en *cualquier* editor ("profe, en el mío se ve bien" es inaceptable).
  - El módulo 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.
- El archivo **debe** 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.
- **Fecha de Entrega.** Viernes 2014-02-28 (Semana 3) hasta las 23:59 VET
- **Valor de Evaluación.** Diez (10) puntos.

Referencias

[1] ASCII Art `http://en.wikipedia.org/wiki/ASCII_art`