Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información.
Laboratorio de Lenguajes de Programación I (CI-3661)
Septiembre - Diciembre, 2010

Proyecto II de Programación Funcional
Dibujando en Funcional

En este proyecto, ud. implementará un lenguaje completo para hacer dibujos en ASCII (muy conocido como arte ASCII [1]). Este proyecto se dividirá en tres partes y este enunciado se actualizará con cada parte publicada.

Parte I: Definición de tipos de datos

En esta primera parte, definiremos los dibujos per se como un tipo abstracto de datos y se trabajarán con algunas operaciones básicas sobre estos.

Tipo de datos `Lienzo`

Primero, definiremos el tipo de datos Lienzo como:

```
data Lienzo = MkLienzo (Integer, Integer) [[Char]]
```

donde el primer argumento de `MkLienzo` es el tamaño del lienzo (alto, ancho), mientras que el segundo corresponde al lienzo per se.

Este tipo de datos debe implementar las siguientes funciones:

- La función:

  ```
  lienzoValido :: Lienzo -> Bool
  ```

  donde `lienzoValido l` representa si un lienzo es válido o no. Un lienzo es válido si su estructura es válida (la lista que representa el lienzo tiene el mismo tamaño indicado en el tipo de datos).

- La función:

  ```
  lienzoVacio :: (Integer, Integer) -> Lienzo
  ```

  donde `lienzoVacio (x, y)` crea un lienzo de altura x y anchura de un solo caracter (por ejemplo, el caracter ASCII 0).

- La función:

  ```
  dibujarPunto :: Lienzo -> Posicion -> Char -> Lienzo
  ```

  con

  ```
  type Posicion = (Integer, Integer)
  ```

  donde `dibujarPunto l pos c` dibuja en la posicion `pos` el caracter `c` en el lienzo `l`.

- La función:

  ```
  dibujarLinea :: Lienzo -> Posicion -> Float -> Integer -> Char -> Lienzo
  ```

  donde `dibujarLinea l pos angulo c` dibuja en el lienzo `l`, partiendo desde `pos`, una línea de longitud `l` y ángulo `angulo`, usando `c`.

- La función:

  ```
  llenar :: Lienzo -> Posicion -> Char -> Lienzo
  ```

donde `llenar l pos c` toma la región delimitada que contiene al punto en `pos` dentro de `l` y llena toda esa región con `c`.

- La función:

  ```
  dibujarCirculo :: Lienzo -> Posicion -> Integer -> Char -> Lienzo
  ```

  donde `dibujarCirculo l pos radio c` dibuja, dentro del lienzo `l`, una circunferencia con centro `pos` y radio `radio` usando `c`.

- La función:

  ```
  obtenerColor :: Lienzo -> Posicion -> Char
  ```

  donde `obtenerColor l pos` es el color ubicado en `pos` dentro del lienzo `l`.

Además, ud. debe hacer que el tipo `Lienzo` sea una instancia de la clase `Show`. Esta instancia no puede ser automática, es decir, ud. debe implementar la función `show` de tal forma que muestre el lienzo de manera correcta. Por ejemplo:

```
Lienzo> MkLienzo (3, 3) [['x', ' ', ' ']. [' ', 'y', 'y'], ['x', ' ', 'y']]
*****
*x  *
* yy*
*x y*
*****
```

Esta parte debe estar en un archivo llamado Lienzo.hs que defina un módulo cuyos unicos componentes a exportar son el tipo `Lienzo` (sin constructor) y las funciones definidas anteriormente.

Parte II: Lenguaje de Dibujo

Para esta segunda parte, ud. va a implementar un pequeño intérprete para un árbol que representa programas de dibujo. Su intérprete va a trabajar con la librería de lienzo ASCII que debe implementar en la parte I.

Ud. debe implementar para esta parte, la función:

```
interpretarComando :: (Lienzo, Char) -> Instruccion -> (Lienzo, Char)
```

donde el tipo `Instruccion` es el siguiente:

```
data Instruccion = Limpiar
                 | EstablecerColor Char
                 | ObtenerColor Posicion
                 | DibujarPunto Posicion
                 | DibujarLinea Posicion Posicion
                 | DibujarCirculo Posicion Magnitud
                 | Llenar Posicion
                 | DibujarCurva Posicion Magnitud Magnitud
                 | DibujarPoligono [Posicion] Bool
                 | DibujarPoligonoRegular Posicion Integer Integer
                 | Triangularizar [Posicion]

type Magnitud = Integer
```

de tal forma que `interpretarComando (l, c) i` es el resultado de ejecutar la instrucción `i` partiendo del lienzo l dibujando con el caracter `c` (el cual llamaremos color activo). Además, debe implementar la función:

```
interpretarBatch :: (Integer, Integer) -> [Instruccion] -> Lienzo
```

tal que `interpretarBatch (x, y) ls` ejecuta las instrucciones de `ls` en secuencia, partiendo de un lienzo vacío de tamaño `(x, y)`.

Especificación de la Ejecución de las Instrucciones

- **Limpieza:** El resultado debe ser un lienzo vacío del tamaño del lienzo original.
- **Establecer color:** Se establece como color activo el caracter dado.
- **Obtener color:** Se establece como color activo el caracter que se encuentra en la posición dada.
- **Dibujar punto:** Se dibuja la posición correspondiente con el color activo.
- **Dibujar lìnea:** Se dibuja una lìnea del punto origen al punto destino con el color activo.
- **Dibujar círculo:** Se dibuja una circunferencia en la posición dada con el radio indicado en la instrucción.
- **Llenado:** Se llena el área indicada en la posición dada con el color activo.
- **Dibujar curva:** Se dibuja una curva (para simplificar, un fragmento de círculo) en la posición y radio dada, con el número de caracteres dados, usando el color activo. El ángulo de partida de esta curva es a los 180 grados.
- **Dibujar polígono:** Hay dos instrucciones: uno para dibujar polígonos no regulares y otro para dibujar aquellos que si sean regulares. Para los polígonos no regulares, se dibuja el polígono usando la lista de posiciones dada en la instrucción. Además, se especifica si dentro del polìgono se entrecruzan las lìneas o no: si no se dibujan las líneas en el orden dado en la lista; si lo es, hay que construir el polígono.

  En el caso que el polígono es regular, se da el punto superior del polígono (izquierdo en el caso de que se requiera la distinción, por ejemplo un cuadrado), el número de lados y la longitud de cada uno. Se construye el polígono en ese punto.
- **Triangularizar:** En esta instrucción se da una lista de puntos, y el objetivo es crear una secuencia de triángulos usando esta lista. Para su implementación va a generar un *strip* de triángulos [2, 3]

Esta parte debe ser entregada en un módulo llamado `Instrucciones.hs`, donde se exporte el tipo con todos sus constructores y las funciones dadas anteriormente.

Parte III: Entrada y Salida

Para esta parte, ud. debe implementar las funciones:

- La función

  ```
  ejecutarPrograma :: FilePath -> IO Lienzo
  ```

  que, dada una dirección de un archivo, devuelve el lienzo resultante de ejecutar el archivo correspondiente. El formato del archivo es el siguiente (la especificaciòn de las instrucciones son iguales a las dadas en la parte 2).

  - **Cabecera:** El archivo debe comenzar con la cabecera `dim x y`, indicando la dimensión del lienzo con x, y enteros.
  - **Establecer color:** `color c`, con c un caracter. Ejemplo: `color x`.
  - **Obtener color:** `get x y`, con x, y enteros. Ejemplo: `get 3 2`
  - **Dibujar punto:** `point x y`. Ejemplo: `point 4 1`
  - **Dibujar lìnea:** `line x1 y1 x2 y2` - con xi, yi enteros. Ejemplo: `line 3 2 4 5` dibuja la lìnea de $(3, 2)$ a $(4, 5)$.
  - **Dibujar círculo:** `circle x y r`, con x, y, r caracter. Ejemplo: `circle 3 1 5` dibuja un cìrculo en posición $(3, 1)$ con radio 5.
  - **Llenado:** `fill x y`. Ejemplo: `fill 2 3`
  - **Dibujar curva:** `curve x y r a`. Ejemplo `curve 3 2 1 50` dibuja una curva con centro $(3, 2)$ de radio 1 y que abarque 50 grados.
  - **Dibujar polígono:** `cpolygon n x1 y1 ... xn yn` para el polígono sin intersecciones, `polygon n x1 y1 ... xn yn` para el libre. `rpolygon x y n l` para el polígono regular.

    Ejemplos: `cpolygon 3 1 1 3 3 3 1`, `polygon 4 1 1 1 3 3 1 3 3`, `rpolygon 5 5 4 10`.
  - **Triangularizar:** `triangularize n x1 y1 ... xn yn`. Ejemplo: `triangularize 4 1 1 1 3 3 1 3 3`.

Y para culminar, el archivo debe tener el comando `end`. Puede asumir que los archivos son correctos. cada instrucción abarca una línea. Un ejemplo de programa:

```
dim 300 200
color y
polygon 5 20 20 20 70 90 30 50 60 40 10
color z
point 1 1
point 1 199
point 299 1
point 299 299
end
```

Todos los puntos comienzan en 0 para esta parte, asi que es valido escribir un punto en la posicion (0, 0).

- La función:

```
exportarLienzo :: Lienzo -> FilePath -> IO ()
```

donde `exportarLienzo l file` genera un archivo de texto en la ubicaciòn `file` que contiene a `l`.

Esta parte debe ser entregada en un archivo llamado `Main.hs`.

Detalles de entrega

- El proyecto debe ser entregado a más tardar el lunes de semana 7 a mas tardar a las 11.59 pm, a su encargado de sección vía correo electrónico. Esta fecha no es negociable. Ud. entregará la implementación de esta parte, más las de las otras dos ese día. No van a haber entregas parciales.
- Ud. debe seguir las reglas de estilo impuestas en el curso (en `http://www.ldc.usb.ve/~caperez/ci3661/sd10/estilo-haskell.txt`).
- Su proyecto debe ser entregado en una carpeta que contenga los primeros apellidos de cada integrante del grupo (p.e. Pérez-Pereira) donde dentro se encuentran los archivos correspondientes a su proyecto. Incluya un archivo léame donde indique con que intérprete usó.
- Este proyecto vale 23 puntos.

Referencias

[1] Arte ASCII. `http://es.wikipedia.org/wiki/Arte_ASCII`
[2] Triangle strip (en inglés). `http://en.wikipedia.org/wiki/Triangle_strip`
[3] `http://www.codercorner.com/Strips.htm`

C. A. Pérez, W. Pereira