

Tipos recursivos y Genericidad

Curso 2003/04, Fecha:14/11/2003

Enunciado 1 (Primos) *Construir un programa que escriba la lista infinita de números primos usando el algoritmo de la criba de Eratóstenes*

Enunciado 2 (ElevationGrid) *El lenguaje VRML (y X3D) incluye una primitiva que permite construir superficies bidimensionales indicando los puntos de elevación. Almacenar el siguiente código VRML en un fichero y visualizarlo.*

```
#VRML V2.0 utf8

Shape {
  geometry ElevationGrid {
    creaseAngle 2.5
    height [ 0 0 0 0 0 0
            0 0 1 1 0 0
            0 1 2 2 1 0
            0 1 2 2 1 0
            0 0 1 1 0 0
            0 0 0 0 0 0 ]
    solid FALSE
    xDimension 6
    zDimension 6
  }
  appearance Appearance {
    material Material { diffuseColor 0 1 0 }
  }
}
```

Enunciado 3 (Plot2D) *El siguiente programa permite representar superficies bidimensionales. Las superficies se representan como funciones de dos variables: **Float** \rightarrow **Float** \rightarrow **Float**.*

Por ejemplo, la función $f(x, y) = \sin(x + y)$ podría visualizarse mediante $w(x, y) \rightarrow \sin(x + y)$

```
> module Pf603 where
```

```

> type Superficie = Float → Float → Float

> w sup = writeFile "fun.wrl"
>         (cabecera ++
>         viewPoint (40,30,95) ++
>         plotS (0,0,0) (0,1,0) sup)

> cabecera = "#VRML_V2.0_utf8\n\n"

> sx, sy :: Float
> sx = 64; sy = 64

> plotS p c = translate p . color c . verS

> viewPoint p = "Viewpoint_{_position_} ++ sh3 p ++ "}\n"

> verS f =
>   "_geometry_ElevationGrid_{_}\n" ++
>   "  _creaseAngle_1.57\n" ++
>   "  _height_{_}\n" ++ puntos f ++ "]\n" ++
>   "  _solid_FALSE\n" ++
>   "  _xDimension_" ++ show (round sx) ++
>   "  _zDimension_" ++ show (round sy) ++ "}"

> puntos f = concat [linea y f ++ "\n" | y ← [1..sy]]
> linea y f = concat [show (f x y) ++ "_" | x ← [1..sx]]

> translate (x,y,z) s =
>   "Transform_{_translation_} ++ sh3 (x,y,z) ++
>   "\n_children_{_} ++ s ++ "}"

> color (r,g,b) s = "Shape_{_}\n" ++ s ++
>   "\n_appearance_Appearance_{_material_Material_{_} ++
>   "\n_diffuseColor_" ++ sh3 (r,g,b) ++ "}"

> sh3 (x,y,z) = show x ++ "_" ++ show y ++ "_" ++ show z ++ "_"

```

Enunciado 4 (superficies) Representar las siguientes funciones de dos variables

- $f(x,y) = \sin(x^2 - y^2)$
- $f(x,y) = 10 * \exp(((x - 32)/32)^2 + ((y - 32)/32)^2)$
- $f(x,y) = 30 * \exp((-2) * (((x - 32)/16)^2 + ((y - 32)/16)^2))$
- $f(x,y) = \text{if } \text{abs}(x - 32) < 10 \ \&\& \ \text{abs}(y - 32) < 10 \ \text{then } 10 \ \text{else } 0$

Enunciado 5 (circulo) Construir una función círculo tal que al evaluar círculo $(x,y) r h s$ genera una superficie insertando un círculo en la posición (x,y) de radio r y altura h a partir de la superficie s

En <http://www.di.uniovi.es/labra/PLF/prac/worlds/circulo.wrl> puede observarse el resultado de evaluar `w (circulo (32,32) 10 3 s1)` Tipo:

```
> circulo :: (Float, Float) -> Float -> Float ->
> Superficie -> Superficie
```

Enunciado 6 (QuadTrees Genéricos) El siguiente fragmento define un tipo de datos que representa quadrees genéricos `QT a` para un tipo `a` cualquiera.

```
> data QT a = B a
>           | D (QT a) (QT a) (QT a) (QT a)
>   deriving Show
```

- Definir varios quadrees cuyos elementos tengan diferentes tipos. Por ejemplo, un quadtree de números flotantes, de cadenas de caracteres, de listas, etc.
- Definir una función `numElem :: QT a -> Int` que calcule el número de elementos de un quadtree
- Definir una función `prof :: QT a -> Int` que calcule la profundidad (o resolución) de un quadtree
- Definir una función `elems :: QT a -> [a]` que devuelva en una lista los elementos de un quadtree
- Definir una función `mapQT :: (a -> b) -> QT a -> QT b` que aplique una función a cada elemento de un quadtree

Enunciado 7 (foldQT) A continuación se define la función `foldQT` que transforma un quadtree en un valor

```
> foldQT :: (b -> b -> b -> b -> b) -> (a -> b) -> QT a -> b
> foldQT f g (B x) = g x
> foldQT f g (D a b c d) = f (foldQT f g a) (foldQT f g b)
>                               (foldQT f g c) (foldQT f g d)
```

Re-escribir las funciones del ejercicio anterior utilizando `foldQT`

Enunciado 8 (q2s) El siguiente fragmento convierte un Quadtree de flotantes en una superficie. Con lo cual es posible visualizar valores de tipo `QT Float`

```
> q2s :: QT Float -> (Float -> Float -> Float)
> q2s qt = q2s' qt ((0,0), d) (\x y -> 0)
> q2s' (B h) ((x,y), d) f =
```

```

> cuadro ((x,y),(x+d,y+d)) h f
> q2s' (D q1 q2 q3 q4) ((x,y),d) f =
>   if d <= 0 then f
>   else
>     let d2 = d 'div' 2
>     in (q2s' q1 ((x,y),d2) .
>         q2s' q2 ((x+d2,y),d2) .
>         q2s' q3 ((x,y+d2),d2) .
>         q2s' q4 ((x+d2,y+d2),d2)) f

> cuadro (p1,p2) h f =
>   \x y → if dentro (x,y) p1 p2
>           then h
>           else f x y

> dentro (x,y) (x1,y1) (x2,y2) =
>   x >= fromInt x1 && x <= fromInt x2 &&
>   y >= fromInt y1 && y <= fromInt y2

> d :: Int
> d = round sx

> e1 = D (B 0) (B 5) (B (-5)) (B 10)

```

Enunciado 9 (colores(Opcional)) Construir un programa que convierta un quadtree de alturas en un quadtree de colores. Cada color puede indicar un nivel de altura.

Enunciado 10 (Tartaglia(Opcional)) Construir un programa que genere el triángulo de Tartaglia

Enunciado 11 (s2q(Opcional)) Construir un programa que convierta una superficie en un quadtree