

Entrada/Salida

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

Enunciado 1 (Par) *A continuación se presenta un programa que pide un número al usuario e indica si el número introducido es par o impar*

```
> module PF703 where
> import Random  -- Para numeros aleatorios
> import Pf603   -- Para ver octrees

> getInt :: String → IO Int
> getInt msg =
>   do
>     putStr msg
>     readLn

> main = do
>   n ← getInt "Número?_"
>   if n `mod` 2 == 0 then putStrLn "Es par"
>   else putStrLn "Es impar"
```

Compilar dicho programa y ejecutarlo

Enunciado 2 (Primo) *Modificar el programa anterior para que pida un número e indique si el número introducido es primo o no*

Enunciado 3 (buscaPrimo) *Modificar el programa anterior para que solicite números de forma repetida hasta que se introduzca un número primo.*

Enunciado 4 (juego) *La generación de números aleatorios requiere efectos laterales. Haskell proporciona la librería **Random** que incluye varias funciones para generación de números aleatorios. Las más habituales podrían ser:*

- **randomIO** :: IO a devuelve un valor aleatorio de tipo a (en realidad, a debe pertenecer a la clase de tipos **Random**)
- **randomRIO** :: (a,a) → IO a devuelve un valor aleatorio dentro de un intervalo de valores.

A modo de ejemplo, el siguiente programa solicita un número n y escribe n números aleatorios entre 0 y 6

```
> aleas :: IO ()
> aleas = do
```

```

>         n ← getInt "Numeros?_"
>         ponAleas n

> ponAleas :: Int → IO ()
> ponAleas 0 = return ()
> ponAleas n = do
>             a ← randomRIO (0,100)
>             putStr "_" ++ show (a :: Int)
>             ponAleas (n - 1)

```

- Modificar el programa anterior para que genere un número aleatorio entre 0 y 100 y luego vaya preguntando al usuario números hasta que el usuario acierte el número generado. En cada iteración, indicar si el número a acertar es mayor o menor que el número introducido.
- Modificar el programa anterior para que al acertar indique cuántos intentos se han realizado

Enunciado 5 (cuentaCar) Escribir un programa Haskell que solicite el nombre de un fichero e indique cuántos caracteres contiene

Enunciado 6 (catch) La función `catch :: IO a → (IOError → IO a) → IO a` funciona de la siguiente forma: `catch ac f` intenta ejecutar la acción `ac`. Si no hay errores devuelve lo que devuelva dicha acción. Si se producen errores, entonces llama a la función `f` pasándole como argumento el error producido.

Por ejemplo, la siguiente función solicita el nombre de un fichero para contar el número de caracteres y, si se produce un error, lo comunica y vuelve a solicitar el fichero.

```

> cuentaSeguro =
>     catch cuentaCar
>     (\e → do
>         putStrLn ("Hubo un error:_" ++ show e)
>         cuentaSeguro)

```

Modificar el juego para que el programa no finalice si el usuario introduce valores no numéricos, sino que continúe pidiendo valores hasta que sean números

Enunciado 7 (octrees Aleatorios(Opcional)) Generar octrees con figuras aleatorias. Pueden emplearse varios procedimientos. Un posible procedimiento tomar un octree y convertir cada elemento básico en una figura aleatoria.

Enunciado 8 (Texto) Añadir el siguiente fragmento que permite generar un mensaje de texto

```

> wt txt = writeFile "text.wrl"
>         (cabecera ++
>          viewPoint (0,0,20) ++
>          putText (0,0,0) (0,0.1,0.1) txt ++

```

```

>         putBox (4, -2, -0.2) (0.8, 1, 1) (10, 10, 0.1)
>     )
>
> putText p c = translate p . color c . verText
> putBox p c = translate p . color c . verBox
> putSphere p c = translate p . color c . verSphere
>
> verText t = "geometry_Text_{_string_}" ++ t ++ "\"}"
>   where l = length t
>
> verBox (sx, sy, sz) =
>   "geometry_Box_{_size_}" ++ sh3 (sx, sy, sz) ++ "}"
>
> verSphere r =
>   "geometry_Sphere_{_radius_}" ++ show r ++ "}"

```

Enunciado 9 (leerTxt) *Modificar el programa anterior para que el mensaje que muestra en el cartel sea leído de un fichero*

Enunciado 10 (enlaces) *El siguiente fragmento permite añadir hiperenlaces entre mundos virtuales*

```

> enlace url desc fig =
>   "Anchor_{_description_}" ++ desc ++
>     "\"_url_[\" ++ url ++
>     \"_]_children_[\" ++ fig ++ "]"
>
> cartel url msg =
>   enlace url ("Enlace_a_" ++ url)
>     (putText (0, 0, 0) (0, 0.1, 0.1) msg ++
>      putBox (4, -2, -0.2) (0.8, 1, 1) (10, 10, 0.1))
>
> mundo1 = cabecera ++
>   viewPoint (0, 0, 20) ++
>   cartel "f2.wrl" "VER_OTRO_MUNDO" ++
>   putSphere (-2, 0, 5) (1, 0, 0) 2
>
> mundo2 = cabecera ++
>   viewPoint (0, 0, 20) ++
>   cartel "p.html" "OTRA_PAGINA"
>
> pagina = "<html><body><a href=\"f1.wrl\">Mundo</a></body></html>"
>
> enlaces =
> do
>   writeFile "f1.wrl" mundo1
>   writeFile "f2.wrl" mundo2
>   writeFile "p.html" pagina

```

Crear varios mundos y páginas Web desde Haskell y enlazarlos. Pueden crearse también mundos que contengan octrees. La instrucción `Inline` de VRML permite insertar un mundo dentro de otro mundo.

Para más información sobre otras instrucciones VRML pueden consultarse las especificaciones. Un tutorial en español puede ser: [Creación de Mundos virtuales en VRML 97](#) de M. J. Abasolo