Haskell Compilado

Usando GHC y la Mónada IO

Compilando

- Compilar es traducir un programa escrito en algún lenguaje de programación a otro lenguaje (en este caso a código que la computadora sabe ejecutar)

- En esta ocasión usaremos GHC que es el compilador del lenguaje Haskell

- El modo de uso es simple:

  ghc --make programa.hs

- Se genera un nuevo archivo ejecutable: “programa” que se puede ejecutar escribiendo:

  ./programa

- Si se desea que el ejecutable tenga un nombre distinto que el archivo que compilamos, se debe hacer:

  ghc --make programa.hs -o nombre_ejecutable

Y… ¿Qué se ejecuta?

- Antes de compilar hay que agregar una función especial a nuestro programa:

```
main :: IO ()
```

- Cuando ejecutamos nuestro programa ya compilado, la función “main” es la única función que se ejecutará.

- La función main debe realizar la tarea que queremos que nuestro programa haga.

- Además debe permitir que el usuario pueda ingresar información.

Entrada y Salida de usuario

- Se denominan operaciones de entrada de datos a la captura de información del mundo exterior. (Por ejemplo la que se obtiene cuando el usuario escribe en el teclado)

- Se denominan operaciones de salida a aquellas que manifiestan información en el mundo exterior (Por ejemplo mostrar algo por pantalla)

- Estas operaciones permiten la interacción del usuario con nuestro programa. Sin ellas nuestro programa sería “autista”.

Programando main :: IO ()

- La función “main” es del tipo “IO ()” que se denomina Mónada IO o mónada de entrada y salida.
- La mónada IO de haskell es una abstracción para manejar la entrada y salida de datos de una función.
- Las funciones para mónadas que usaremos son: getLine, readLn, putStrLn y print.

Funciones de la mónada IO

- readLn :: Read a => IO a:

  Lee la entrada del usuario por teclado y la transforma en un tipo conocido de Haskell (ya sea una lista, cadena, booleano, etc)

- getLine :: IO String

  Lee por teclado la entrada del usuario y la transforma en una cadena.

- putStrLn :: String -> IO ()

  Muestra el contenido de una cadena por la pantalla

- print :: Show a => a -> IO ()

  Muestra un elemento del tipo “a” por pantalla (siempre y cuando este en la clase Show)

Notación do {}

- Una de las formas de definir funciones del tipo mónada IO es usando la notación do {}

- Como nosotros solo usaremos esta notación para definir la función main :: IO () podemos valernos de ciertas reglas simples:

  do $\{e_0;e_1;e_2;\ldots;e_n\}$

  * Cada uno de las expresiones e_i pueden ser de dos especies:

    -Entrada:

    Por ejemplo: xs <- readLn
    s <- getLine

    -Salida:

    Por ejemplo: print [1,2,3]
    putStrLn “Hola”
    print s
    print xs

  * La última expresión e_n debe ser de salida

Ejemplos

- El clásico hola mundo:

```
main :: IO ()
main = do
       putStrLn "Hola mundo!!"
```

- Un programa para sumar:

```
main :: IO ()
main = do
       putStrLn "Ingrese un numero";
       x <- readLn;
       putStrLn "Ingrese otro numero";
       y <- readLn;
       putStrLn "El resultado es";
       print (x + y)
```

Ejemplos

- Usando la función “todosPares” del proyecto 1:

```
todosPares :: [Int] -> Bool
todosPares [] = True
todosPares (x:xs) = even x && todosPares xs


main :: IO ()
main = do
       putStrLn "Ingrese una lista: ";
       xs <- readLn;
       print (todosPares xs)
```

Ejemplos

- Una prueba del modulo Data

```
import Data

main :: IO ()
main = do
        putStrLn "Escriba una cadena:";
        s <- getLine;
        putStrLn"La longitud del dato es: ";
        print (data_length (data_fromString s))
```