PROGRAMANDO EN HASKELL

Capítulo 2 - Primeros Pasos

Glasgow Haskell Compiler

- GHC es la implementación líder de Haskell, conformada por un compilador y un intérprete;
- La naturaleza interactiva del intérprete lo adecua para enseñar y prototipar;
- GHC está disponible gratuitamente en:

```
www.haskell.org/downloads
```

Iniciando GHCi

El intérprete puede ser iniciado desde la consola de comandos, con solo escribir ghci:

```
$ ghci

GHCi, version X: http://www.haskell.org/ghc/  :? for help

Prelude>
```

El prompt de GHCi es >, y significa que el intérprete está listo para evaluar una expresión.

Por ejemplo, puede ser usado como calculadora para evaluar expresiones numéricas simples:

```
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

Preludio

Haskell viene con una gran librería de funciones estándar (The Standard Prelude). Sumadas a las funciones numéricas familiares tales como + y *, la librería también provee muchas funciones útiles sobre listas.

- Selecciona el primer elemento de una lista:

```
> head [1,2,3,4,5]
1
```

- Borra el primer elemento de una lista:

```
> tail [1,2,3,4,5]
[2,3,4,5]
```

- Seleccionar el n-avo elemento de una lista:

```
> [1,2,3,4,5] !! 2
3
```

- Tomar los primeros n elementos de una lista:

```
> take 3 [1,2,3,4,5]
[1,2,3]
```

- Borrar los primeros n elementos de una lista:

```
> drop 3 [1,2,3,4,5]
[4,5]
```

- Calcular el tamaño de una lista:

```
> length [1,2,3,4,5]
5
```

- Calcular la suma de una lista de números:

```
> sum [1,2,3,4,5]
15
```

- Calcular el producto de una lista de numeros:

```
> product [1,2,3,4,5]
120
```

- Concatenar dos listas:

```
> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- Invertir (“Voltear”) una lista:

```
> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

Aplicación de Funciones.

En matemáticas, la aplicación de funciones se denota con paréntesis, y la multiplicación se denota con yuxtaposición o espacio.

```
f(a,b) + c d
```

Aplicar la función f a `a` y a `b`, y añadir el resultado al producto de `c` y `d`.

En Haskell, la aplicación de funciones se denota usando espacio, y la multiplicación se denota usando *.

```
f a b + c*d
```

Lo anterior, pero usando sintaxis de Haskell.

Más aún, se asume que la aplicación de funciones tiene mayor precedencia que todas las otras operaciones.

```
f a + b
```

Significa (f a) + b, en lugar de f (a + b).

Ejemplos

Matemáticas	Haskell
`f(x)`	`f x`
`f(x,y)`	`f x y`
`f(g(x))`	`f (g x)`
`f(x,g(y))`	`f x (g y)`
`f(x)g(y)`	`f x * g y`

Scripts de Haskell

- Al igual que las funciones en la librería estándar, Tú también puedes definir tus propias funciones;

- Las nuevas funciones se definen en un script, un archivo de texto constituido por una secuencia de definiciones;

- Convención, los scripts de Haskell tienen la extensión de archivo .hs, no es obligatorio, pero es útil para propósitos de identificación.

Mi Primer Script

Al desarrollar un script de Haskell, es útil tener dos ventanas abiertas, una ejecutando un editor para el script, y la otra ejecutando GHCi.

Inicia un editor de texto, y escribe las siguientes dos definiciones de función, guarda el script como test.hs:

```
double x = x + x

quadruple x = double (double x)
```

Deja el editor abierto, y en otra ventana inicia GHCi con el nuevo script:

```
$ ghci test.hs
```

Ahora la librería estándar de funciones y el archivo test.hs han sido cargados, se pueden usar las funciones de ambos orígenes:

```
> quadruple 10
40

> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Deja GHCi en ejecución, regresa al editor, añade las siguientes dos definiciones, guarda los cambios:

```
factorial n = product [1..n]

average ns = sum ns `div` length ns
```

Nota:

- div está encerrado con comillas invertidas;
- x `f` y es azúcar sintáctica para f x y.

GHCi no detecta automaticamente que el script ha sido cambiado, por lo cual se debe ejecutar un comando de recarga antes de utilizar las nuevas definificiones:

```
> :reload
Reading file "test.hs"

> factorial 10
3628800

> average [1,2,3,4,5]
3
```

Comandos Útiles en GHCi

Comando	Significado
:load *nombre*	carga script *nombre*
:reload	recarga el script
:set editor *n*	fija el editor a *n*
:edit *nombre*	edita el script *nombre*
:edit	edita el script actual
:type *expr*	ver el tipo de *expr*
:?	muestra ayuda
:quit	salir de GHCi

Requirementos para Nombres

- Los nombres de Función y argumento deben comenzar con una letra minúscula. Por ejemplo:

```
myFun    fun1    arg_2    x'
```

- Convención, los argumentos que son listas, usualmente tienen un sufijo s en su nombre. Por ejemplo:

```
xs    ns    nss
```

La Regla de Diseño

En una secuencia de definiciones, cada definición debe comenzar precisamente en la misma columna:

```
a = 10
b = 20
c = 30
```

```
a = 10
 b = 20
c = 30
```

```
 a = 10
b = 20
 c = 30
```

La regla de diseño evita la necesidad de sintaxis explícita para indicar el agrupamiento de definiciones.

```
a = b + c
    where
      b = 1
      c = 2
d = a * 2
```


```
a = b + c
    where
      {b = 1;
       c = 2}
d = a * 2
```

Agrupamiento implícito

Agrupamiento explícito

Ejercicios

(1) Usa GHCi con las diapositivas 2-7 y 13-16.

(2) Corrige los errores de sintaxis en el siguiente programa, y prueba tu solución en GHCi.

```
N = a 'div' length xs
    where
       a = 10
      xs = [1,2,3,4,5]
```

(3) Muestra como la función last (En Prelude) que selecciona el último elemento de una lista puede ser definido usando las funciones presentadas en este capítulo.

(4) ¿Se te ocurre otra posible definición?

(5) De manera similar, muestra como la función init (En Prelude) que borra el último elemento de una lista puede ser definido en dos formas diferentes.

Fuente.

Profesor Graham Hutton.
http://www.cs.nott.ac.uk/~pszgmh/

Traducción al español.

Caleb Josue Ruiz Torres.
https://www.calebjosue.com