

Tecnológico de Costa Rica

Tarea Programada 1

Asignatura: Lenguajes de Programación

Estudiante: Josué Suárez Campos – 2016089518

Profesor: José Araya Monge

2018

## Introducción

La primera tarea programa del curso Lenguajes de Programación consiste en implementar en el lenguaje de programación funcional Haskell un sistema que permita el cálculo de variables dependientes de otras. Para ello se espera que el usuario ingrese ecuaciones que permitan el cálculo de variables a partir de los valores de otras. Por ejemplo:

$$\begin{aligned}x &= 2 + w \\ y &= x - z \\ r &= w * t\end{aligned}$$

A partir de las ecuaciones previamente ingresadas el sistema establecerá un árbol para cada expresión, el cual irá cambiando según se vayan ingresando ecuaciones que contengan variables relacionadas.

Mediante comandos ingresados por el usuario para la asignación de valores a los diferentes términos se retornará un valor numérico resultante de la ecuación.

## Estructuras de datos usadas

Las estructuras de datos utilizadas en la tarea programada son los que se presentan en la siguiente imagen:

```
type Operacion = Int -> Int -> Int
data Termino = Variable String | Entero Int
data Arbol = Hoja Termino | Nodo Operacion Arbol Arbol
type Estado = [(String, [String], Arbol, [String], Arbol)]
```

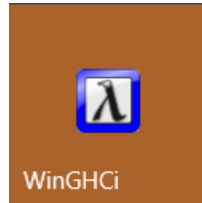
*Figura 1. Estructuras de datos.*

La estructura principal es la de Árbol, el cual es utilizado para establecer las ecuaciones. Está compuesto de nodos, los cuales contienen la operación matemática, además también se tienen las hojas que almacenan las variables o números de la ecuación. Dichas hojas utilizan el dato Término que realiza la distinción de variable y un número.

Además del árbol y sus componentes, se encuentra el Estado que permite el almacenamiento de las diferentes ecuaciones junto con toda su información correspondiente: [(Variable, Lista de variables de la ecuación, Árbol de la ecuación, Lista de variables de la ecuación modificada, Árbol de la ecuación modificada)]. El Estado como estructura de datos no sigue un orden específico, se van agregando nuevas tuplas a la izquierda.


## Instrucciones de ejecución

Primeramente, se debe ingresar al programa llamado WinGHCi (Figura 2), el cual permitirá el acceso al sistema.



*Figura 2. Software WinGHCi*



Seguidamente presione el botón de abrir archivo  y seleccione el archivo llamado **Proyecto-2016089518.hs** el cual se encuentra en el directorio **TP1-Suarez-Campos-Josue**. Una vez abierto el archivo Haskell, escriba en la línea de comandos la palabra **main** (Figura 2).

```
*Main> main  
>> |
```

*Figura 3. Ingreso al sistema*

A continuación, se prosigue mediante el uso de diferentes comandos con diferentes funcionalidades:

- Ingreso de ecuaciones (**ie**): Para el ingreso de ecuaciones debe utilizar el comando **ie** seguidamente debe expresar la ecuación que sea agregar, las

operaciones que soporta el sistema son sumas (+), restas (-) y multiplicaciones (\*). Dichas ecuaciones deben cumplir el siguiente formato:

```
<ecuacion> ::= <variable> = <termino> <op> termino>
<termino>  ::= <variable> | <entero>
<variable> ::= Secuencia alfanumérica que empieza con letra
<entero>   ::= Secuencia con uno o más dígitos
<op>       ::= + | - | *
```

*Figura 4. Formato de ecuación*

Además, para que estas sean efectivas deben cumplir las siguientes condiciones:

- La variable a la izquierda de la ecuación no debe tener una ecuación previa. Es decir, si se ingresa la ecuación:  $x = 2 + w$ , x solo debe aparecer una vez como incógnita con una operación de términos anterior.
- La variable a la izquierda de la ecuación no debe aparecer a la derecha de esa misma ecuación. Es decir, ecuaciones como  $x = 2 + x$ , no son aceptadas por el sistema.
- La nueva ecuación no debe formar un ciclo de dependencias con las ecuaciones anteriores.

Por lo tanto, cumpliendo lo anterior, un ejemplo de uso del comando **ie** es el siguiente:

```
>> ie y = a + b  
{y, [a, b], (a + b), [a, b], (a + b)}
```

*Figura 5. Comando ie*

Como se puede apreciar en la figura anterior, cuando se añade una nueva ecuación el sistema retorna la información relacionada con la variable indicada. Dicha información corresponde a **{Variable, Lista variables de la ecuación original, Representación del árbol de la variable, Lista de variables de la ecuación modificada a partir de otras ecuaciones, Representación del árbol de la variable modificada}**. En caso de que la ecuación ingresada presente un error, el sistema la denegará y retornará un mensaje de error.

- **Mostrar variable (mv):** Mediante el presente comando el sistema mostrará la información asociada con la variable definida en el comando. Es importante tener previamente creada la variable para poder mostrarla. La sintaxis de dicho comando es la siguiente:

```
>> mv y  
{ y, [a, b], a + b, [a, c], a + (2 + c) }
```

*Figura 6. Comando mv*

Como se puede apreciar en la imagen anterior, se indica la variable que sea desea mostrar y seguidamente el sistema retornará la información correspondiente.

- **Mostrar ambiente (ma):** El comando **ma** muestra toda la información relacionada a cada una de las variables que han sido ingresadas hasta el momento. Dicho comando no conlleva parámetros, basta con la digitación del comando **ma**.

```
>> ma
{ y, [a, b], a + b, [a, c], a + (2 + c) }
{ x, [y, z], y * z, [a, c, z], (a + (2 + c)) * z }
{ w, [z, v], z - v, [z, v], z - v }
{ b, [c], 2 + c, [c], 2 + c }
```

*Figura 7. Comando ma*

- **Calcular variable (cv):** La funcionalidad que presenta el comando cv es calcular el valor numérico de una ecuación ingresada previamente, mediante la introducción de valores numéricos asociados a los términos variables que son utilizados en la ecuación. En el comando se indica la variable y los valores numéricos que serán utilizados en la ecuación, el orden de estos está relacionado con el mostrado de información de una variable.

```
>> cv y 5 3      a=5, c=3
7                Esto es: 5+(2+3)
```

*Figura 8. Comando cv*

- Calcular variable original (**cvo**): El comando cvo sigue la misma funcionalidad y sintaxis del comando anterior, la diferencia radica en que el comando es utilizado para el cálculo de la variable original, es decir, la resolución de la ecuación sin modificaciones.

```
>> cvo x 5 2      y=5, z=3
15               Esto es: 5 * 3
```

*Figura 9. Comando cvo*

- Mostrar parámetros (**mp**): Mediante el comando mp se muestra todos los parámetros utilizados para el cálculo de las ecuaciones ingresadas por el usuario. La sintaxis de este comando es solamente la escritura de mp.

```
>> mp
a c z v
```

*Figura 10. Comando mp*

- Evaluar todo (**et**): El comando et permite la resolución de todas las ecuaciones ingresadas por el usuario. Para ello el usuario debe ingresar los valores numéricos de los parámetros de los que el sistema depende siguiendo el orden mostrado por el comando mp. La sintaxis y el retorno del sistema a dicho comando se muestran a continuación.



```
>> et 5 3 4 1      Esto es: a=5 c=3 z=4 v=1
[ (y,7) , (x,40) , (w,3) , (b,5) ]
```

Figura 11. Comando et

- Terminar (**fin**): Como último comando se encuentra el comando fin cuyo único propósito es terminar la ejecución del sistema.

### Corridas ejemplo

```
>> ie x1 = y + z
{x1, [y, z], (y + z), [y, z], (y + z)}
>> ie x1 = 1 + n
Ocurrió un error! La variable x1 se encuentra previamente definida.
>> ie x = x1 * w
{x, [x1, w], (x1 * w), [y, z, w], ((y + z) * w)}
>> mv x
{x, [x1, w], (x1 * w), [y, z, w], ((y + z) * w)}
>> cv x 2 3 4
20
>> ie u = w - t
{u, [w, t], (w - t), [w, t], (w - t)}
>> mp
[y, z, w, t]
>> et 2 3 4 1
(x1, 5), (x, 20), (u, 3)]
>> ie z = a * b
{z, [a, b], (a * b), [a, b], (a * b)}
```

```

>> ma
{x1, [y, z], (y + z), [y, a, b], (y + (a * b))}
{x, [x1, w], (x1 * w), [y, a, b, w], ((y + (a * b)) * w)}
{u, [w, t], (w - t), [w, t], (w - t)}
{z, [a, b], (a * b), [a, b], (a * b)}

>> cv x 2 3 4 5
70
>> ie w = 2 * u
Error! La ecuación ingresada genera un bucle
>> ie w = 2 * z
{w, [z], (2 * z), [a, b], (2 * (a * b))}
>> ma
{x1, [y, z], (y + z), [y, a, b], (y + (a * b))}
{x, [x1, w], (x1 * w), [y, a, b], ((y + (a * b)) * (2 * (a * b)))}
{u, [w, t], (w - t), [a, b, t], ((2 * (a * b)) - t)}
{z, [a, b], (a * b), [a, b], (a * b)}
{w, [z], (2 * z), [a, b], (2 * (a * b))}

>> mp
[y, a, b, t]
>> et 2 3 4 5
(x1, 14), (x, 336), (u, 19), (z, 12), (w, 24)]

>> ie u = r * s
Ocurrió un error! La variable u se encuentra previamente definida.

```

## **Comentarios**

Se cumplió con todo lo requerido para el funcionamiento correcto del sistema solicitado. No se presentan errores o limitaciones en la ejecución de la tarea programada.