### INSTITUTO TECNOLÓGICO DE COSTA RICA I SEMESTRE 2018

**INGENIERÍA EN COMPUTACIÓN**

**IC-4700 LENGUAJES DE PROGRAMACIÓN**

#### TAREA PROGRAMADA I

La tarea consiste en implementar en Haskell un mecanismo para recalcular los valores de variables que dependen de otras variables.

Se tendrán ecuaciones que establecen la forma de calcular el valor de una variable como función de otras. Por ejemplo:

**x = 2 + w**

**y = x - z**

**r = w \* t**

Las ecuaciones son de la forma:

**<ecuacion> ::= <variable> = <termino> <op> termino>**

**<termino> ::= <variable> | <entero>**

**<variable> ::=** *Secuencia alfanumérica que empieza con letra*

**<entero> ::=** *Secuencia con uno o más digitos*

**<op> ::= +** | **-** | **\***

El programa va recibiendo de entrada una ecuación a la vez, y las va acumulando.

Como no se trata de resolver un sistema de ecuaciones matemáticas, una variable solo puede aparecer a lo sumo una vez en el lado izquierdo de una igualdad. De modo que el siguiente caso es inválido :

**r = w + t**

**r = 2 \* w**

Además, la variable que aparece al lado izquierdo no puede aparecer al lado derecho:

**n = r \* n**

Tampoco se puede dar la formación de ciclos de dependencias entre variables porque de nuevo eso es plantear un sistema de ecuaciones matemáticas. En el siguiente caso la tercera ecuación forma el siguiente ciclo de dependencias: **r** depende de **t** que depende de **s** que depende de **r**:

**r = w + t**

**t = w \* s**

**s = q + r**

En otras palabras las dependencias entre variables deben formar un grafo dirigido acíclico.

Conforme se van introduciendo nuevas ecuaciones se deben ir recalculando cuáles variables se necesitan para el cálculo y sus fórmulas correspondientes.

Por ejemplo, después de introducir la ecuación **y=a+b** se tiene la información siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| variable | parámetros originales | función original | parámetros vigentes | función vigente |
| **y** | **a, b** | **a + b** | **a, b** | **a + b** |

Si ahora se introduce la ecuación **x=y\*z** se debe utilizar la información ya almacenada sobre **y**, para cambiar la fórmula a **x=(a+b)\*z**, ya que **y** aparece en la ecuación de **x**. Esto se especifica cambiando el conjunto de parámetros y la ecuación y llamándolos ***vigentes***:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| variable | parámetros originales | ecuación original | parámetros vigentes | ecuación vigente |
| **y** | **a, b** | **a + b** | **a, b** | **a + b** |
| **x** | **y, z** | **y \* z** | **~~y, z~~**  **a, b, z** | **~~y \* z~~**  **(a + b) \* z** |

Si se introduce la ecuación **w=z-v**, esta no afecta ni es afectada por las ecuaciones anteriores:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| variable | parámetros originales | ecuación original | parámetros vigentes | ecuación vigente |
| **y** | **a, b** | **a + b** | **a, b** | **a + b** |
| **x** | **y, z** | **y \* z** | **a, b, z** | **(a + b) \* z** |
| **w** | **z, v** | **z - v** | **z, v** | **z - v** |

Si se introduce la ecuación **b=2+c** s, **b** no es afectada por **x**,**y**,**w**, (ninguna de ellas aparece en la ecuación de **b**); por otro lado **x**,**y** son afectadas por **b** (ya que **b** aparece en sus ecuaciones vigentes). Se debe usar la información de **b** para actualizar los parámetros y fórmulas vigentes de **x**,**y**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| variable | parámetros originales | ecuación original | parámetros vigentes | ecuación vigente |
| **y** | **a, b** | **a + b** | **~~a, b~~**  **a, c** | **~~a + b~~**  **a + (2 + c)** |
| **x** | **y, z** | **y \* z** | **~~a, b, z~~**  **a, c, z** | **~~(a + b) \* z~~**  **(a + (2 + c)) \* z** |
| **w** | **z, v** | **z - v** | **z, v** | **z – v** |
| **b** | **c** | **2 + c** | **c** | **2 + c** |

**Operaciones**

El programa deberá proveer las siguientes operaciones:

Insertar-ecuacion (**ie**):

* introduce una nueva ecuación en el ambiente
* verificar que no se dé ninguna de las siguientes situaciones; en caso de darse alguna, deberá emitir un mensaje de advertencia e ignorar la nueva ecuación
  + la variable a la izquierda de la ecuación no debe tener una ecuación previa
  + la variable a la izquierda de la ecuación no debe aparecer a la derecha de esa misma ecuación
  + la nueva ecuación no debe formar un ciclo de dependencias con las ecuaciones anteriores; esto es para cada una de las ecuaciones anteriores, revisar que la variable de la nueva ecuación no aparece a la derecha de una ecuación anterior y que la variable de esa ecuación anterior no aparece a la derecha de la nueva ecuación; **para esta revisión debe usar las ecuaciones vigentes**
* la ecuación se recibirá como una tira de texto que sigue el formato especificado arriba
* rechazar la ecuación si tiene errores sintácticos en ***ecuación***
* como resultado se debe mostrar la información creada al almacenar dicha ecuación

**>> ie y = a + b**

**{ y, [a, b], a + b, [a, b], a + b }**

**>> ie x = y \* z**

**{ x, [y, z], y \* z, [a, b, z], (a + b) \* z }**

**>> ie w = z - v**

**{ w, [z, v], z - v, [z, v], z - v }**

**>> ie b = 2 + c**

**{ b, [c], 2 + c, [c], 2 + c }**

Mostrar-variable (**mv**):

* dada una variable debe mostrar la información asociada con esa variable; en particular se muestra el conjunto de parámetros originales de esa variable y el conjunto de parámetros vigentes

**>> mv y**

**{ y, [a, b], a + b, [a, c], a + (2 + c)}**

**>> mv x**

**{ x, [y, z], y \* z, [a, c, z], (a + (2 + c)) \* z }**

Mostrar-ambiente (**ma**):

* mostrar la información asociada con todas las variables que se han definido

**>> 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 }**

Calcular-variable (**cv**):

* dada una variable y un conjunto de valores correspondientes a sus parámetros **vigentes**, debe evaluar la ecuación que calcula dicha variable dados esos parámetros

**>> cv y 5 3** *a=5, c=3*

**7** *Esto es: 5+(2+3)*

**>> cv x 5 3 4** *a=5, c=3, z=2*

**40** *Esto es: ( 5 + (2+3) )\* 4*

Calcular-variable-original (**cvo**):

* como el anterior caso, solo que deben usarse los parámetros **originales**

**>> cvo x 5 2 y=***5, z=3*

**15** *Esto es: 5 \* 3*

Mostrar-parámetros (**mp**):

* devuelve la lista de parámetros que se requieren para calcular todas las variables
* en caso de querer usar el siguiente comando *evaluar-todo*, se deben especificar los valores siguiendo el orden señalado en el primer elemento

**>> mp**

**a c z v**

Evaluar-todo (**et**):

* usa los valores especificados para los parámetros en el orden mostrado por la operación anterior, *mostrar-parámetros*, y evalúa todas las variables definidas por ecuaciones

**>> et 5 3 4 1** *Esto es: a=5 c=3 z=4 v=1*

**[ (y,7), (x,40), (w,3), (b,5) ]**

Terminar (**fin**)

* Termina el ciclo de ejecución.

### Sugerencias

Las siguientes son algunas acciones que pueden ser útiles para desarrollar la tarea. No es obligatorio seguirlas y la funcionalidad puede ser programada de otra manera.

### Definir un tipo de datos Arbol que permita almacenar las expresions: en las hojas vienen constantes o el nombre de una variable, y en los nodos internos viene una función binaria de enteros (+,-,\*) y los dos subárboles.

### Elabore una función crearArbol que tome una tira de caracteres con una operación binaria simple ("*operando operación operando*") y devuelva un árbol como del punto 1: crearArbol :: String -> Arbol crearArbol "x + 3" crearArbol "w \* z"

### Elabore una función sustVar que tome una variable y un Arbol que representa su ecuación y sustituya en otro Arbol las apariciones de esa variable por copias del Arbol asociado.: sustVar :: Variable -> Arbol -> Arbol -> Arbol -> [String] sustVar *b* *<Árbol para (2+c)>* *<Árbol para ((a+b)\*z)>* produce *<Árbol para ((a+(2+c))\*z)>*

### Elabore una función listaVar que tome un Arbol y devuelva una lista con las variables que aparecen en dicho árbol; cada variable debe aparecer una sola vez en la lista: listaVar :: Arbol -> [String] listarVar (crearArbol "x + 3") produce ["x"] listarVar *<Árbol para (a+(2+c))\*z>* produce ["a","c","z"]

### Elabore una función evalArb que tome un Arbol y una lista de valores, y devuelva el resultado de evaluar dicho Arbol usando esos valores; los valores se asocian con las variables siguiendo el orden especificado por el resultado de listaVar para ese Arbol: evalArb :: Arbol -> [Int] -> Int evalArb *<Árbol para (a+(2+c))\*z>* [3,1,5] produce (3+(2+1))\*5 = 30 porque el orden de los parámetros dado por listaVar es ["a","c","z"], de modo que a=3, c=1, z=5. Una alternativa es que la lista de valores ya venga asociada con los nombres: evalArb *<Árbol para (a+(2+c))\*z>* [("a",3),("c",1),( "z",5)]

### Ajustar el programa *ciclo interactivo.hs* que se les adjunta para implementar la funcionalidad pedida.