Lenguajes de programación 2019-1
Práctica 2: Paradigma imperativo

El objetivo es extender el paradigma funcional al estilo imperativo implementando secuencias de comandos, efectos sobre la memoria y ciclos.

Dada la gramática:

$$x \mid n \mid \mathsf{true} \mid \mathsf{false} \mid e + e \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \mid \mathsf{iszero}\ e \mid \mathsf{let}\ x = e\ \mathsf{in}\ e\ \mathsf{end} \mid e < e \mid e = e \mid \neg e$$

con $n \in \mathbb{Z}$, la extendemos con las expresiones:

$$\ell_n \mid e_1 := e_2 \mid \mathsf{ref}\ e \mid !e \mid e_1; e_2 \mid \mathsf{while}\ e_1\ \mathsf{do}\ e_2 \mid ()$$

que denotan asignación, alojamiento, obtención, secuenciación, ciclos y el valor unit, respectivamente.

La respectiva extensión en Haskell es la siguiente:

```
data EAB = Var String
         | VNum Int
         | VBool Bool
         | Suma EAB EAB
         | Ifte EAB EAB EAB
         | Iszero EAB
         | Let String EAB EAB
         | Menor EAB EAB
         | Eq EAB EAB
         | Neg EAB
         | L Int
         | Asig EAB EAB
         | Ref EAB
         | Deref EAB
         | Seq EAB EAB
         | While EAB EAB
         | Unit deriving (Show, Eq)
```

con L el constructor para las direcciones de memoria, la cual representaremos con el tipo:

```
type Mem = [(LDir,Val)]
```

donde *LDir* es un alias del tipo *EAB* que servirá para identificar expresiones de la forma $L\ n$. Esto es para hacer énfasis en la idea de que una dirección la representamos con el símbolo ℓ. *Val* también es un alias del tipo *EAB* que se restringe a los contructores que representan valores, esto es para enfatizar que una memoria guarda tales valores. Por lo que una memoria tiene la forma `[(`$L\ n_1$`,`v_1`),...,(`$L\ n_k$`,`v_k`)]`.

1. Semántica dinámica (Entrega: 5 de octubre del 2018)

Implemente las siguientes funciones que tienen efecto sobre memoria.

- `accessMem::LDir ->Mem->Maybe Val`

  Función que recibe una etiqueta de la forma $(L\ n)$, y devuelve el valor alojado en memoria en la posición n.

  - `>accessMem (L 1) [(L 0,VBool False),(L 1,VNum 3)]`
    `Just (VNum 3)`
  - `>accessMem (L 2) [(L 0,VBool False),(L 1,VNum 3)]`
    `Nothing`

- `update::LDir->Val->Mem->Mem`

  Actualiza la celda de una memoria con un valor dado. En caso de que la dirección de memoria no se encuentre en la memoria, dicha dirección se agrega.

  - `>update (L 3) (VNum 4) [(L 0,VBool True)]`
    `[(L 0,VBool True),(L 3,VNum 4)]`
  - `>update (L 0) (VBool False) [(L 0,VBool True),(L 1,VNum 3)]`
    `[(L 0,VBool False),(L 1,VNum 3)]`

- `eval1::(Mem,EAB)->(Mem,EAB)`

  Un paso de evaluación en el paradigma imperativo.

  ```
  >eval1 ([(L 1, VNum 4)],Suma (VNum 2) (VNum 3))
  ([(L 1,VNum 4)],VNum 5)
  ```

- `evals::(Mem,EAB)->(Mem,EAB)`

  Varios pasos de evaluació en el paradigma imperativo.

  ```
  >evals ([(L 1, VNum 4)], Prod (Suma (VNum 2) (VNum 3)) (Deref $ L 1))
  ([(L 1,VNum 4)],VNum 20)
  ```

- `interp::EAB->EAB`

  Función que interpreta un expresión comenzando con una memoria vacía.

  ```
  >interp $ Let "x" (Ref $ VNum 3) (Suma (VNum 4) (Deref $ Var "x"))
  VNum 7
  ```

Adicionalmente, deben dar una implementación imperativa, usando los términos del lenguaje *EAB* definidos arriba, de una función que diga si un entero es par, del factorial, de la división, del residuo de una división, del máximo común divisor y una función que simule el *swap*, es decir, que intercambie el contenido de dos referencias. Sus funciones deben ser ejecutadas por la función *interp*.

2. Semántica estática (Entrega: 12 de octubre del 2018)

Para la verificación de tipos con referencias, los contextos son de la forma:

$$\Gamma \mid \Sigma \vdash e : \mathsf{T}$$

donde Γ es un conjunto cuyos elementos son de la forma $x : \mathsf{T}$, y Σ es un conjunto cuyos elementos son de la forma $\ell : \mathsf{T}$. En Haskell, representaremos los contextos y los tipos así:

```
type Ctx = ([(String,Tipo)],[(LDir,Tipo)])
data Tipo = TInt | TBool deriving (Show,Eq)
```

Implemente la función `vt::Ctx->EAB->Tipo`, que realiza la verificación de tipos de una expresión *EAB* siguiendo las reglas de tipado para el paradigma imperativo.

```
> vt([],[]) $ Let "x" (Ref $ VNum 3) (Suma (VNum 4) (Deref $ Var "x"))
TInt
```

Además, deberán incluir cinco expresiones de prueba que corran con la función *vt*, dos deben fallar y tres deben estar correctamente tipadas. Deben usar todos los constructores de *EAB*.