Universidad del Istmo de Guatemala
Facultad de Ingenieria
Ing. en Sistemas
Informatica 1
Prof. Ernesto Rodriguez - erodriguez@unis.edu.gt

Hoja de trabajo #6

Fecha de entrega: 6 de Septiembre, 2018 - 11:59pm

Instrucciones: Resolver cada uno de los ejercicios siguiendo sus respectivas instrucciones. El trabajo debe ser entregado a traves de Github, en su repositorio del curso, colocado en una carpeta llamada "Hoja de trabajo 6". Al menos que la pregunta indique diferente, todas las respuestas a preguntas escritas deben presentarse en un documento formato pdf, el cual haya sido generado mediante Latex.

Nota: En este deber se omitira la ubicación exacta del compilador de elm, y solo se escribira elm. Por ejemplo, en vez de escribir:

> `node_modules\elm repl`

Se escribira:

> `elm repl`

Adicionalmente, asegurarse que las funciones y modulos que sean declarados en su deber correspondan exactamente a los nombres escritos en dicho deber ya que se utilizaran pruebas automatizadas para calificar.

Ejercicio #1 (25%)

Los **numeros naturales unarios** se pueden definir de la siguiente manera:

$$\textbf{type}\ \texttt{Natural}\ =\ \texttt{Suc Natural}\ |\ \texttt{Cero}$$

En donde `Cero` corresponde al numero cero (0) y `Suc` es el constructor que construye un natural representandolo como el sucesor de otro numero. Ejercicio:

- Definir en Elm la función "`resta : Natural → Natural → Natural`" la cual calcula la resta entre dos naturales como estan definidos anteriormente. Si, el resultado fuese a ser negativo, retornar cero.
- Definir la función "`multiplicacion : Natural → Natural → Natural`" la cula multiplica dos naturales como fueron definidos anteriormente.
- Definir la función "`division : Natural → Natural → (Natural, Natural)`" la cual debe calcular la division y el residuo resultante de dividir un natural dentro del otro.

Ejercicio #2 (25%)

Definir el tipo `Expresion` para representar **expressiones matematicas** en Elm. Una expresion matematica esta compuesta de los siguientes casos:

- `Valor`: El cual debe aceptar un entero (`Int`) como parametro.
- `Suma`: El cual debe aceptar dos expresiones como parametro.
- `Mult`: El cual debe aceptar dos expresiones como parametro.

Por ejemplo, la expression "3+8∗5∗2+4" seria representada (respetando las reglas de procedencia de la suma y multiplicación) como "`Suma` (`Valor` 3) (`Suma` (`Mult` (`Valor` 8) (`Mult` (`Valor` 5) (`Valor` 2))) (`Valor` 4))"

Ejercicio #3 (50%)

Definir una función llamada "`parsear : string` → `Maybe Expresion`" que toma una expresión representada como un `string` y produce una expresión respetando la procedencia de operaciones. Se recomienda seguir los siguientes consejos:

- Generalize el tipo `Expresion` creando un nuevo tipo llamado `GExpresion` el cual acepta un parametro y utiliza ese parametro en vez de un `Int`. Luego puede definir el tipo *mathttExpresion* asi:
  > **type alias** `Expresion` = `GExpresion Int`
- Crear un tipo nuevo llamado `Estado`, con dos constructores. Un constructor acepta un `Int` y el otro un `ListChar`. La idea es que el primer caso representa un valor ya producido mientras que la lista de caracteres representa un valor que aun necesita procesamiento.
- El tipo estado se utilizara en las expresiones para facilitar la conversion. El algoritmo es el siguiente:
  1. Empezar con un solo `Valor`, en el cual coloca un `Estado` con la expresión entera.
  2. Buscar en el `Estado` el operador de menor procedencia. ie. la suma más a la derecha
  3. Selecciónar el constructor adecuado en base a esa operación (`Suma` o `Mult`)
  4. Construir los valores de ese constructor llamando recursivamente a la función.

Para su implementación solo debe considerar operaciónes bien formadas que no contengan espacios. Si la operación no cumple estos criterios, puede retornar `Nothing`