Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Traductores e Interpretadores (CI-3725)
Abril-Julio, 2011

Proyecto Único - Etapa 1
Análisis Lexicográfico

Esta primera etapa del proyecto consiste en implementar un módulo de análisis lexicográfico del interpretador del lenguaje Vectorinox que se quiere construir. Específicamente se desea que Ud. implemente el módulo en cuestión utilizando las expresiones regulares que provee el lenguaje de programación Ruby o usando la herramienta de implementación de analizadores lexicográficos Alex (el cuál usa el lenguaje de programación Haskell),

Los *tokens* relevantes del lenguaje serán:

- Cada una de las palabras reservadas usadas en el lenguaje Vectorinox.
- Los identificadores de variables y funciones en Vectorinox. A diferencia de las palabras reservadas, los identificadores corresponden a un único *token* `TkId`, que contiene el nombre del identificador asociado.
- Cada uno de los símbolos que denotan separadores u operaciones especiales en Vectorinox.
- Los números reales, correspondientes a un único *token* `TkNum`. Este *token* siempre tendrà asociado el valor numérico partícular asociado.
- Los *strings*, correspondientes a un *token* `TkString`, que tenga asociado el contenido del string correspondiente.
- Los caracteres (o secuencias de éstos) que representen los operadores definidos por Vectorinox.
- Los espacios en blanco, tabuladores, saltos de línea y comentarios deben ser ignorados.
- Las diferencias entre mayúsculas y minúsculas deben preservarse.

Su analizador lexicográfico debe ser ejecutado mediante el comando `vecti`, y el resultado debe ser una secuencia de *tokens*, con su contenido (en el caso que lo haya) y su posición. Por ejemplo:

```
$ vecti prueba.vec
TkId a (Linea 1, Columna 3)
TkNum 3.4 (Linea 1, Columna 5)
TkIgual (Linea 2, Columna 1)
…
```

Además, el comando `vecti` debe de contener una opción para facilitar un programa en consola. Dicha ejecución debe realizar el análisis lexicográfico sobre esa entrada. Por ejemplo:

```
$ vecti -e "print 'Hola mundo!'"
TkPrint (Linea 1, Columna 1)
TkString Hola Mundo! (Linea 1, Columna 7)
```

En el caso que hayan errores léxicos en el lenguaje, Ud. debe mostrar **todos** los errores que ocurran. En el caso que ocurra error, no se puede imprimir *tokens* reconocidos anteriormente. Es **inaceptable** el manejo de errores mediante *tokens*. Dichos mensajes de error deben indicar la posición en el que dicho error ocurrió. Por ejemplo (asumiendo que `prueba2.vec` contenga un error):

```
$ vecti prueba2.vec
ERROR: Caracter no esperado '¡' en línea 2, columna 1.
… (resto de mensajes de error)
```

Obligaciones en Ruby

- Ud. debe modelar sus *tokens* mediante una clase `Token`, el cuál todos los *tokens* que Ud. defina son subclases de ésta.
- Debe implementar una clase `Lexer`, tal que:
    - Tiene dos constructres de instancias: uno que reciba un objeto de la clase `IO::File` de Ruby, previamente abierto, asociado al archivo que contiene el texto a procesar; el segundo recibe un *string*.
    - Un metodo publico de nombre `Lexer.yylex()` que cada vez que es invocado.
        - Retorna un objeto de la jerarquía de clases `Token` particular al siguiente token reconocido.
        - Retorna el valor `nil` si se alcanza el final del archivo.
        - Produce una excepción cuando se detecta un error durante el procesamiento (e.g. comentarios anidados, símbolo inesperado).
- Su implementación debe realizarse usando expresiones regulares.
- Ud. debe usar la herramienta RDoc para generar la documentación de su código fuente.

Obligaciones en Haskell

- Ud. debe modelar sus *tokens* mediante un tipo algebraico `Token`. Dicho tipo algebraico debe de contener un constructor por cada token en el lenguaje.
- Ud. debe de implementar un analizador lexicográfico usando la herramienta Alex.
- Ud. debe usar la herramienta Haddock para generar la documentación de su código fuente.

Entrega de Implementación

Ud. debe de entregar un archivo `.tar.gz` (**no** puede ser ni `.zip` ni `.rar`) la cuál, al descomprimirse, debe generar una carpeta con su número de grupo (por ejemplo, `G16`) cuyo contenido está estructurado de la siguiente forma:

- Una carpeta `src` que contenga todos sus archivos fuente. Al menos debe tener un archivo que contenga las definiciones de los tokens (`Tokens.rb` o `Tokens.hs`) y un archivo que contenga su analizador lexicográfico (`Lexer.rb` o `Lexer.x`).
- Una carpeta `bin` (vacía al momento de entrega) que, al completar el proceso de compilación, debe de tener el archivo `vecti` con todos los archivos necesarios para su ejecución. Dicha carpeta debe de funcionar en cualquier sitio y sin depender del contenido de este paquete.
- Una carpeta `doc` (vacía al momento de entrega) que, en el caso que se le solicite a las herramientas de compilación, contenga la documentación generada de su proyecto.
- Un `Makefile` que contenga las siguientes opciones:
    - Una opción de compilación que realice los pasos de compilación necesarios y genere los resultados en la carpeta `bin`. Esta es la opción por defecto (es decir, ejecutando `make` solamente). Aunque Ruby es un lenguaje interpretado, recuerde que debe respetar las restricciones indicadas.
    - Una opción que permite generar la documentación. Esta opción debe ser invocada mediante `make doc`.
    - Una opción que permita borrar los archivos creados en el paso por defecto. Esta opción se invoca mediante `make clean`.
    - Una opción que borre todos los archivos generados. Esta opción se invoca mediante `make cleanall`.
- Un `README` con información de interés para el proyecto. Debe contener los integrantes del equipo y la funcionalidad no implementada.

La documentación de su proyecto debe ser generada a partir de su código fuente. Por lo tanto, su entrega **no** debe tener indicio alguno de documentación. Además, **todos** los fuentes de su proyecto deben de estar completa y correctamente documentados.

Detalles de entrega

- **Fecha de Entrega:** Domingo 22 de mayo de 2011 (Semana 4) a las 11.59 am. Todo proyecto entregado después de esta hora tendrá ponderación de cero (0) puntos.
- **Valor de Evaluación:** Cinco (5) puntos.