Universidad Simón Bolívar
Dpto. de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Abril-Julio 2009

Proyecto Unico - Entrega 1
Análisis Lexicográfico

Esta primera etapa del proyecto corresponde al módulo de análisis lexicográfico del interpretador del lenguaje YISIEL que queremos construir. Específicamente, se desea que Ud. implemente el módulo en cuestión utilizando las expresiones regulares del lenguaje de programación Ruby, además de hacer una breve revisión de los conceptos y métodos teórico-prácticos relevantes.

El analizador lexicográfico a ser construido en esta etapa deberá ser encapsulado en la clase `Lexer`. Los *tokens* deben ser modelados como una jerarquía de clases Ruby a partir de la clase general `Token` y utilizando una subclase de ésta por cada tipo de *token* particular del lenguaje. Si se reciben caracteres que no corresponden a ningún *token* del lenguaje, debe producirse un mensaje de error; es **inaceptable** modelar los errores con un *token*. Tanto los *tokens* reconocidos como los mensajes de error deben acompañarse de la posición dentro del archivo (línea y columna) en la cual han sido encontrados.

Los *tokens* relevantes serán:

- Cada una de las palabras reservadas utilizadas en la sintaxis de YISIEL.
- Los identificadores de variables y procedimientos en YISIEL. A diferencia de las palabras reservadas, estos identificadores corresponderán a un único *token* `TkId`, que tendrá asociado como atributo el identificador particular reconocido.
- Cada uno de los símbolos que denotan separadores u operaciones especiales en YISIEL.
- Los números enteros, que corresponderán a un *token* llamado `TkNum`. Este *token* siempre tendrá asociado como atributo al valor numérico particular reconocido.
- Los caracteres o secuencias de caracteres que corresponden a los operadores matemáticos y lógicos en YISIEL.
- Los espacios en blanco, tabuladores, saltos de línea y los comentarios deben ser ignorados.
- La diferencia entre minúsculas y mayúsculas debe preservarse.

Así, una interacción con el analizador lexicográfico sería similar a (asumiendo que el archivo `test.ysl` existe y contiene los elementos léxicos del lenguaje descrito)

```
$ yisiel.rb test.ysl
TkId foo (Linea 1, Columna 3)
TkNum 42 (Linea 1, Columna 9)
TkMenor (Linea 2, Columna 1)
TkStr esta es la cadena encontrada (Linea 2, Columna 5)
...
```

Entrega de la Implementación

Ud. debe entregar un archivo `.tar.gz` o `.tar.bz2` (**no** puede ser `.rar`, ni `.zip`) que al expandirse genere un directorio con el nombre de su grupo (e.g. `G42`) dentro del cual encontrar:

- El archivo `Token.rb` conteniendo el código fuente Ruby para implantar las clases que representarán a los tokens. Debe aplicar las técnicas de herencia y polimorfismo para la representación de los tokens y el manejo de los métodos de acceso. Concretamente, los atributos deben ser "sólo lectura" y establecidos al momento de la creación de cada instancia.
- El archivo `Lexer.rb` conteniendo el código fuente Ruby para implantar la clase `Lexer` que ofrece la funcionalidad del analizador lexicográfico. La clase debe proveer:
  - El constructor de instancias que recibe como argumento un objeto de la clase `IO::File` de Ruby, previamente abierto, asociado al archivo que contiene el texto a procesar.
  - Un método público 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).
- El archivo `yisiel.rb` conteniendo el código fuente Ruby para el programa principal. Este archivo será utilizado con Ruby 1.8 en Debian GNU/Linux sin modificación alguna; los profesores **no** vamos a editar archivos en modo alguno, de manera que Ud. debe verificar que el archivo que envía funciona perfectamente con la versión de Ruby incluida en Debian GNU/Linux. Haga la prueba en el LDC para estar seguro.
- Los módulos **deben** estar completa y correctamente documentados utilizando la herramienta RDoc. Ud. **no** entregará los documentos HTML generados, sino que deben poder **generarse** de manera automática incluyendo acentos y símbolos especiales. Es **inaceptable** que la documentación tenga errores ortográficos.
- El programa principal será utilizado desde la línea de comandos de dos maneras diferentes:
  - Si se invoca el programa suministrado **exactamente** un argumento en la línea de comandos, el programa debe intentar abrir ese archivo y procesarlo. El procesamiento debe retornar **todos** los *tokens* encontrados y **todos** los errores lexicográficos detectados, si existieran, e.g.

    ```
    $ yisiel.rb test.ysl
    ... lista de tokens y errores entremezclados
    ```

    Si el archivo suministrado como argumento no existe o no puede abrirse por la razón que sea, el programa debe terminar indicando el problema. Es **inaceptable** que el programa

- Si se invoca el programa **sin** argumentos en la línea de comandos, el programada debe ofrecer un *prompt* en el cual escribir el nombre del archivo a procesar. El procesamiento debe retornar **todos** los *tokens* encontrados y **todos** los errores lexicográficos detectados, si existieran, e.g.

  ```
  $ yisiel.rb
  Archivo a interpretar: test.ysl
  ... lista de tokens y errores entremezclados
  ```

- En ambos casos, los errores lexicográficos deben ser reportados de manera clara y acompañados de la información de línea y columna, e.g.

  ```
  Caracter inesperado '@' encontrado en linea 21, columna 42.
  ```

- Un archivo PDF con el desarrollo de las preguntas de la siguiente **Revisión Teórico-Práctica**:

  1. Presente tres expresiones regulares[1] E_1, E_2 y E_3 que correspondan respectivamente al reconocimiento de la palabra clave `as`, de la palabra clave `array` y del identificador de variables.
  2. Presente los diagramas de transición de tres autómatas finitos (posiblemente no-determinísticos) M_1, M_2 y M_3 que reconozcan respectivamente a los lenguajes denotados por E_1, E_2 y E_3.
  3. Presente un autómata finito (posiblemente no-determinístico) que reconozca la unión de los lenguajes $L(M_1)$, $L(M_2)$ y $L(M_3)$.
  4. Note que, a efectos de implementar un analizador lexicográfico, es importante que el autómata M sepa reportar a cuál de los tres lenguajes pertenece cada palabra que él reconozca. Esto significa que M debe poder identificar a cuál de los tres lenguajes corresponde cada estado final. ¿Puede esto crear algún conflicto? En caso afirmativo, ¿cómo debe resolverse este conflicto?
  5. Construya un autómata finito determinístico mínimo N equivalente a M. Indique a cuál de los tres lenguajes corresponde cada estado final justificando su respuesta.

- **Fecha de Entrega.** Lunes 2008-05-18 (Semana 5) hasta las 15:30
- **Valor de Evaluación.** Seis (6) puntos.

[1] Use la notación formal estudiada en teoría. La notación de expresiones regulares de Ruby no es aceptable en éste caso.