Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3725 - Traductores e Interpretadores
Abril - Julio 2018

Analizador Lexicografico

Aquí debemos realizar un análisis lexicográfico del interpretador del lenguaje. Es decir, crear o generar analizadores lexicográficos (generar tokens; generar errores estáticos).

Como entrada, debemos aceptar cualquier secuencia de caracteres, y como salida, los *tokens* relevantes reconocidos. Si no se reconocen los caracteres como parte de nuestro lenguaje, entonces debemos arrojar un error. Todo *token* debe ir acompañado de su número de fila y columna.

- Las palabras reservadas o claves son las correspondientes a nuestro lenguaje: begin, while, bool, if, etc. En este caso, los tokens deben llamarse: `Tk⟨Palabra Clave⟩`, donde ⟨Palabra Clave⟩ es la palabra clave a la que representa el token, con su primera letra en mayúscula. Por ejemplo, para la palabra clave begin, su token sería TkBegin.
- Los identificadores de variables. Todos los identificadores corresponden a un *token* llamado *TkId*. Este token siempre tendrá asociado como atributo el identificador reconocido. Ejemplo: al leer: `contador`, se dará como salido `TkId("contador")`.
- Los literales numéricos son secuencias no-vacías de dígitos decimales. Ellos son agrupados bajo el token `TkNum`. Por ejemplo: `TkNum(30)`.
- Los literales booleanos, están representados por dos tokens (para `True` y `False`). El usado para `True` será: `TkTrue` y el asociado a `False` sera: `TkFalse`.
- Los literales para caracteres, los cuales serán cadenas de un símbolo envuelto en comillas simples (o dos símbolos, si se trata de un caractér escapado). De manera análoga a los identificadores y los literales numéricos, estos serán agrupados bajo el token `TkCaracter`. Este token tendrá como atributo el caracter reconocido, por ejemplo `TkCaracter('p')`.
- Cada uno de los símbolos que denotan separadores, los cuales se presentan a continuación:
  - `","` - `TkComa`
  - `"."` - `TkPunto`
  - `":"` - `TkDosPuntos`
  - `"("` - `TkParAbre`
  - `")"` - `TkParCierra`
  - `"["` - `TkCorcheteAbre` `"]"` - `TkCorcheteCierra`
  - `"{"` - `TkLlaveAbre`

- "}" – TkLlaveCierra "->" – TkHacer
- "<-" – TkAsignacion

- Los símbolos que denotan a operadores aritméticos, booleanos, relacionales, o de otro tipo se presentan a continuación:
  - "+" – TkSuma
  - "-" – TkResta
  - "*" – TkMult
  - "/" – TkDiv
  - "%" – TkMod
  - "/\" – TkConjuncion
  - "\/" – TkDisyuncion
  - "not" – TkNegacion
  - "<" – TkMenor
  - "<=" – TkMenorIgual
  - ">" – TkMayor
  - ">=" – TkMayorIgual
  - "=" – TkIgual
  - "++" – TkSiguienteCar
  - "--" – TkAnteriorCar
  - "#" – TkValorAscii
  - "::" – TkConcatenacion
  - "$" – TkShift

Espacios en blanco, tabuladores y saltos de línea deben ser ignorados. Cualquier otro carácter deberá ser reportado como error.

A continuación un ejemplo de una posible entrada. Lo que recibe el analizador lexicográfico y luego la secuencia de tokens de salida, todos acompañados de *dos números* (el primero de ellos la fila en donde se encuentra el token y el segundo la columna):

```
with
  var contador : int

begin
   contador <- 35.

End
```

```
TkWith 1 1,
TkVar 2 5, TkId("contador") 2 9, TkDosPuntos 2 18, TkInt 2 20,
TkBegin 4 1,
TkId("contador") 7 5, TkAsignacion 7 14, TkNum(35) 7 17, TkPunto 7 19
TkEnd 8 1
```

El analizador lexicográfico solo es capaz de reconocer secuencias ***arbitrarias*** de tokens, a pesar de que esas secuencias pueden estar sintácticamente incorrectas!.

En cuanto a los errores; los únicos errores detectables a nivel lexicográfico corresponden a caracteres incorrectos o mal formados. Por ejemplo:

```
with
        var contador : !int

   begin
        contador? <- 35.

end
      Error: Caracter inesperado "!" en la fila 2, columna 20
      Error: Caracter inesperado "?" en la fila 7, columna 13
```

Su analizador lexicográfico debe reportar todos los errores léxicos, en caso de haberlos. Cuando un error es encontrado, los tokens se hacen irrelevantes (ya que no corresponden a un programa correcto), por lo que no deben ser mostrados.

A continuación se explica cuáles son los lenguajes y herramientas permitidas, los detalles de la entrega de la implementación y finalmente, en qué consiste la revisión teórico-práctica.
Lenguajes y Herramientas Permitidos Para la implementación de este proyecto se cuenta para escoger con ocho (8) lenguajes de programación. Estos son:

- **Python**: Lenguaje de scripting, orientado a objetos. Es posible conseguir Python desde la siguiente dirección Web:
  (http://www.python.org/download/).
  Para Python la herramienta generadora de analizadores lexicográficos a utilizar es a la vez la misma que genera analizadores sintácticos. Por lo tanto, deberán manejarse algunas nociones de gramáticas libres de contexto antes de tiempo para poder trabajar con la misma. Esta herramienta se llama PLY y puede ser encontrada desde la siguiente dirección Web:
  (http://www.dabeaz.com/ply/).

- **Ruby:** Lenguaje de scripting, orientado a objetos. Es posible conseguir Ruby desde la siguiente dirección Web:
  (http://www.ruby-lang.org/en/downloads/).
  En el caso particular de Ruby no hay una buena herramienta generadora de analizadores lexi- cográficos, por lo que el trabajo deberá hacerse manualmente a través de las expresiones regulares que provee el lenguaje.
- **Haskell:** Lenguaje funcional puro. Es posible conseguir Haskell desde la siguiente dirección Web:
  (http://www.haskell.org/ghc/download.html/).
  Si decide utilizar Haskell, su implementación deberá ser compatible con el compilador proporcionado (GHC).

Para Haskell la herramienta generadora de analizadores lexicográficos a utilizar se llama Alexy puede ser encontrada en la siguiente dirección Web:
(http://www.haskell.org/alex/).

- **Java:** Lenguaje imperativo, orientado a objetos. Es recomendable utilizar versiones de Javaiguales o posteriores a la 1.5, dependiendo también de los requisitos de las herramientas a utilizar. Es posible conseguir Java desde la siguiente dirección Web:
  (http://www.java.com/es/download/).
  Para Java, la herramienta generadora de analizadores lexicográficos a utilizar se llama JFlex y puede ser encontrada en la siguiente dirección Web:
  (http://jflex.de/download.html/).

Detalles de entrega

Le tienen que mandar todo a Fernando Lovera: flovera1@gmail.com independientemente de si lo haces en Haskell, Python, Ruby o Java!. Pongan como subject: ci3725-`<Apellidos>`, en donde Apellidos se refiere a los apellidos de los integrantes del grupo.

El correo debe contener el código fuente en el lenguaje y la herramienta de su elección, entre los permitidos, de su analizador lexicográfico. Todo debe estar debidamente documentado! (esto es importante). Para ejecutar el analizador lexicográfico, se debe utilizar este comando:
"`./Lex <Archivo>`" (Esto también es muy importante), por lo que es posible que se tenga que incorporar un script que permita que la llamada a su programa se realice de esta forma. `<Archivo>` tendrá el programa escrito en nuestro lenguaje, que será analizado lexicográficamente. Deben abstenerse de imprimir nada más que lo pedido y vigilar la salida de su programa para que corresponda con lo pedido.

También deben explicar cómo está implementado su código (al estilo de un informe muy sencillo), para que puedan justificar su implementación.

Fecha de Entrega: Jueves 17 de Mayo (Semana 4), hasta las 12:00 pm.
Valor: 5 puntos.

Ferderico Glaviani; Fernando Lovera; Genesis Kufatty; / Abril 2018