Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 – Traductores e Interpretadores
Abril – Julio 2013

Etapa I
Análisis Lexicográfico

Esta entrega consiste en la implementación de un analizador lexicográfico para el lenguaje RangeX, el cual constituye la primera fase del interpretador que se desea construir a lo largo del trimestre. Basado en la definición del lenguaje, disponible en la página oficial del laboratorio, deberá identificar los *tokens* relevantes, formular expresiones regulares que los reconozcan e implantar el analizador utilizando la herramienta generadora escogida.

Entre los *tokens* relevantes del lenguaje se encuentran:

- Cada palabra reservada del lenguaje RangeX.
- Los identificadores de variables y constantes. Debe ser un único *token* cuyo contenido sea el nombre del identificador asociado.
- Cada símbolo utilizado para denotar operadores y separadores.
- Los números enteros. Debe ser un único *token* cuyo contenido sea el número reconocido.
- Las cadenas de caracteres encerradas entre comillas. Debe ser un único *token* cuyo contenido sea la cadena leída.
- Las constantes booleanas.

Otras consideraciones:

- Los espacios en blanco, tabuladores, saltos de línea y comentarios deben ser ignorados. Es inaceptable manejarlos como *tokens*.
- Se debe preservar la diferencia entre mayúsculas y minúsculas.

Su programa principal debe llamarse `rangex` y recibirá como primer argumento el nombre del archivo a analizar. La salida debe mostrar todos los tokens reconocidos de manera legible e incluyendo para cada uno la **línea** y **columna** donde se encontró. Por ejemplo:

```
$ rangex archivo.rgx
```

Suponiendo que el contenido del archivo `archivo.rgx` es:

```
program
 begin
  declare
   a, b, c as int;
   d, e, f as range
  a = b + 3;
  read e
  //Esto es un comentario.  Debe ser ignorado.
end
```

Cuya salida debe ser la representación de un *token* por línea:

```
TkProgram (Línea 1, Columna 1)
TkBegin (Línea 2, Columna 1)
TkDeclare (Línea 3, Columna 1)
TkId "a" (Línea 4, Columna 1)
TkComma (Línea 4, Columna 2)
TkId "b" (Línea 4, Columna 3)
TkComma (Línea 4, Columna 4)
TkId "c" (Línea 4, Columna 5)
TkAs (Línea 4, Columna 6)
TkInt (Línea 4, Columna 8)
TkSemicolon (Línea 4, Columna 11)
TkId "d" (Línea 5, Columna 1)
TkComma (Línea 5, Columna 2)
TkId "e" (Línea 5, Columna 3)
TkComma (Línea 5, Columna 4)
TkId "f" (Línea 5, Columna 5)
TkAs (Línea 5, Columna 6)
TkRange (Línea 5, Columna 8)
TkSemicolon (Línea 5, Columna 13)
TkId "a" (Línea 6, Columna 1)
TkEqual (Línea 6, Columna 2)
TkId "b" (Línea 6, Columna 3)
TkPlus (Línea 6, Columna 4)
TkNum "3" (Línea 6, Columna 5)
TkSemicolon (Línea 6, Columna 6)
TkRead (Línea 7, Columna 1)
TkId "b" (Línea 7, Columna 5)
TkEnd (Línea 9, Columna 1)
```

La salida anterior es una representación aproximada. Lo importante es que se represente un *token* por línea en la cual se imprimen: *token*, lexema, línea y columna.

En caso de encontrar errores léxicos, debe mostrar **todos** y **sólo** los errores encontrados. La salida no puede tener una mezcla de *tokens* reconocidos y errores léxicos. Es inaceptable que se manejen los errores como *tokens*. Por ejemplo:

Suponiendo que el contenido del archivo `archivo.rgx` es:

```
program
 begiñ
  declare
   a, b, c as int;
   d, e, f as range
  a := b + 3;
  read e
  //Esto es un comentario.  Debe ser ignorado.
end
```

La salida debería ser:

```
Error:  caracter inesperado "ñ" en línea 2, columna 5.
Error:  caracter inesperado ":" en línea 6, columna 2.
```

Es importante notar que cada *token* producido por su analizador lexicográfico debe corresponder a un tipo de datos y no simplemente ser impreso a la salida estándar. Esto a modo de poder suministrarlos luego como información para el analizador sintáctico. De esta forma, usted deberá implementar un programa principal que invoque al analizador lexicográfico y sea luego el que imprima a la salida estándar la representación como texto de los *tokens* encontrados.

Lenguajes y Herramientas a usar Para la implementación de este proyecto se cuenta para escoger con tres (3) lenguajes de programación. El lenguaje que elija ahora será el que deberá usar para todas las entregas. No podrá cambiarlo después. Estos son:

- *Python:*
  - Interpretador *python* 2.6.6.
  - Generador de analizadores lexicográficos y sintácticos *PLY* 3.3.3. Note que para esta entrega solo le interesa utilizar el submódulo **lex** de PLY, puede ignorar por ahora todo lo relacionado con **yacc**.
- *Haskell:*
  - Compilador *ghc* 6.12.1.
  - Generador de analizadores lexicográficos *Alex* 2.3.3.
- *Ruby:*
  - Interpretador *ruby* 1.8.7.
  - En el caso particular de *Ruby* no hay una buena herramienta generadora de analizadores lexicográficos, por lo que el trabajo deberá hacerse manualmente a través de las expresiones regulares que provee el lenguaje.

Entrega de la Implementación

- Un correo electrónico, a todos los preparadores, con el código fuente de su analizador lexicográfico. Todo el código debe estar debidamente documentado.
- Incluya un Makefile si va a utilizar Haskell. Si su entrega no compila no será corregido.
- Un *breve* informe (`README.txt`) explicando la formulación/implementación/problemas de su analizador lexicográfico y justificando todo aquello que usted considere necesario.
- Respete las reglas de juego expuestas en la página oficial del curso.

Nota: Es importante que su código pueda ejecutarse en las máquinas del LDC, pues es ahí y únicamente ahí donde se realizará su corrección.

Referencia Bibliográfica

[WM95] R. Wilhelm & D. Maurer. *Compiler Design.* Addison-Wesley, 1995.

[S97] T. Sudkamp. *Languages and Machines.* Second Edition. Addison-Wesley, 1997.

Fecha de Entrega: Viernes 10 de Mayo (Semana 3), hasta las 11:59:59 pm.
Valor: 5%.
