Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3725 - Traductores e Interpretadores

Abril - Junio 2015

Lanscii - Etapa I

Análisis Lexicográfico (4%)

Especificación de la entrega

En la primera etapa de desarrollo del interpretador para el lenguaje `Lanscii` se debe implementar su analizador lexicográfico y realizar un breve análisis de los conceptos y métodos teórico-prácticos utilizados durante esta primera fase del proyecto.

Siguiendo las especificaciones de la definición del lenguaje deberán identificar los *tokens* relevantes, crear expresiones regulares que los reconozcan e implantar el analizador utilizando la herramienta escogida por el equipo de trabajo. Este analizador lexicográfico recibirá como entrada cualquier secuencia de caracteres e imprimirá los *tokens* reconocidos del lenguaje `Lanscii`. En caso de encontrar caracteres que no corresponden a ningún *token* relevante, se deberá imprimir un mensaje de error.

Deben ser considerados los siguientes como *tokens* de `Lanscii`:

- Las palabras reservadas del lenguaje `Lanscii`.
- Los identificadores de cada variable, recordando que este *token* es único y definido por el nombre del identificador asociado.
- Los valores constantes permitidos por el lenguaje `Lanscii`:
  - Los números enteros, cuyo contenido del *token* asociado sea el número reconocido.
  - Las constantes booleanas: `true` y `false`.
  - Las constantes lienzos, tomando como contenido del *token* asociado el símbolo envuelto entre `<` y `>`; e.g. el literal lienzo `<|>` se representa como `TkLienzo("|")`.
- Los símbolos utilizados para especificar la operación de asignación `:=`; los operadores aritméticos, booleanos y sobre lienzos; separadores y bloques.

Recuerden que las tabulaciones, los espacios en blanco, los saltos de línea y los comentarios **no** se definen como *tokens* del lenguaje, por lo tanto **no** deben de ser reconocidos por el analizador lexicográfico.

Ejecución

Para la ejecución del interpretador su programa deberá llamarse `lanscii` y recibirá como primer argumento el nombre del archivo con la cadena de caracteres a analizar. Note que el archivo de entrada *puede no contener* un código correcto de `Lanscii`, el analizador lexicográfico únicamente se encargará de reconocer los *tokens* relevantes del lenguaje en la cadena especificada en él.

Por salida estándar, se debe mostrar todos y cada uno de los *tokens* reconocidos por el analizador, especificando -en cada uno- la **línea** y **columna** donde fue encontrado.

Si un código suministrado posee errores léxicos: *tokens* no relevantes al lenguaje `Lanscii`, se debe mostrar por salida estándar **todos** los errores encontrados y **sólo** los errores encontrados. Es decir, la salida **no** puede contener tanto *tokens* reconocidos como errores léxicos del código. Asimismo, deben recordar que los errores léxicos no son considerados como *tokens* reconocidos por el lenguaje.

En el caso que sea encontrado un error de cierre de sección de comentarios, esto es: una sección de comentarios abierta y no cerrada hasta alcanzar el final del archivo, se debe reportar por salida estándar **únicamente este error**, no deben ser mostrados los *tokens* encontrados o los errores lexicográficos, en caso de existir.

Ejemplo de un programa correcto en `Lanscii` y su salida de análisis lexicográfico respectiva:

```
{- example program -}

{ @cake lie !GLaDOS |
    read lie;
    cake = <|>;
    (cake = lie ? GLaDOS = true : glaDos = false);
    write -1
}
```

Salida:

```
./lanscii example.lc
token LCURLY value ({) at line: 3, column: 1
token AT value (@) at line: 3, column: 3
token IDENTIFIER value (cake) at line: 3, column: 4
token IDENTIFIER value (lie) at line: 3, column: 9
token EXCLAMATION MARK value (!) at line: 3, column: 13
token IDENTIFIER value (GLaDOS) at line: 3, column: 14
token PIPE value (|) at line: 3, column: 21
token READ value (read) at line: 4, column: 5
```

```
token IDENTIFIER value (lie) at line: 4, column: 10
token SEMICOLON value (;) at line: 4, column: 13
token IDENTIFIER value (cake) at line: 5, column: 5
token EQUALS value (=) at line: 5, column: 10
token CANVAS value (|) at line: 5, column: 12
token SEMICOLON value (;) at line: 5, column: 15
token LPARENTHESIS value (() at line: 6, column: 5
token IDENTIFIER value (cake) at line: 6, column: 6
token EQUALS value (=) at line: 6, column: 11
token IDENTIFIER value (lie) at line: 6, column: 13
token QUESTIONMARK value (?) at line: 6, column: 17
token IDENTIFIER value (GLaDOS) at line: 6, column: 19
token EQUALS value (=) at line: 6, column: 26
token TRUE value (true) at line: 6, column: 28
token COLON value (:) at line: 6, column: 33
token IDENTIFIER value (glaDos) at line: 6, column: 35
token EQUALS value (=) at line: 6, column: 42
token FALSE value (false) at line: 6, column: 44
token RPARENTHESIS value ()) at line: 6, column: 49
token SEMICOLON value (;) at line: 6, column: 50
token WRITE value (write) at line: 7, column: 5
token MINUS value (-) at line: 7, column: 11
token NUMBER value (1) at line: 7, column: 12
token RCURLY value (}) at line: 8, column: 1
```

Otro ejemplo:

```
{{{=||
```

Salida:

```
token LCURLY value ({) at line: 1, column: 1
token LCURLY value ({) at line: 1, column: 2
token LCURLY value ({) at line: 1, column: 3
token EQUALS value (=) at line: 1, column: 4
token PIPE value (|) at line: 1, column: 5
token PIPE value (|) at line: 1, column: 6
```

Note que el programa ejemplo es sintácticamente incorrecto, lo que muestra que para esta entrega sólo se deben de reconocer los *tokens* relevantes al lenguaje. Por otro lado, un ejemplo de programa incorrecto con su salida asociada:

```
{ %bad sym |
    & = bad+sym;
    write \;
}
```

Salida:

```
Error: Unexpected character: "&" at line: 2, column: 5
Error: Unexpected character: "\" at line: 3, column: 11
```

Otro ejemplo:

```
{ %n |
    read n;
    write n-1;
}
{- example program
```

Salida:

```
./lanscii bad_example2.lc
Error: Comment section opened but not closed at line: 5, column: 1
```

Note que para la salida de un código *correcto* en `Lanscii`, en esta primera etapa de desarrollo, se imprime el toda la información interesante del *token*: el identificador del reconocido, su valor asociado y su posición (línea y columna). Recuerden que la correctitud de cada entrega determinará el desarrollo de la siguiente. Todas las entregas pertenecen a un mismo proyecto de desarrollo.

Implementación

Para la implementación del interpretador del lenguaje `Lanscii`, pueden escoger uno (1) de los cuatro (4) lenguajes de programación a continuación. Para cada uno de ellos se indica las herramientas disponibles para el desarrollo de un interpretador de código:

- *C++*:
  - *g++* 4.7
  - *Flex* y *Bison*. Para este etapa del proyecto se utilizará la herramienta de creación de analizadores lexicográficos: *Flex*. Entregas futuras requerirán de *Bison* para el análisis sintáctico.
- *Java*:
  - *javac* 1.6 ó 1.8
  - *JFlex* y *CUP*. *JFlex* es la herramienta dedicada a la creación de analizadores lexicográficos en *Java*, por lo cual será la correspondiente a utilizar en esta etapa de desarrollo. Entregas posteriores utilizarán *CUP* para el análisis sintáctico.

- *Ruby:*
  - *ruby* 1.9 ó 2.1
  - *Rexical* y *Racc*. *Rexical* es la herramienta dedicada a la creación de analizadores lexicográficos en *Ruby*, **el uso de *Rexical* es opcional**. Entregas posteriores utilizarán _Racc para el análisis sintáctico.
- *Haskell:*
  - *GHC* 7.6.3 ó 7.8.3
  - *Alex* y *Happy* . Para esta etapa de desarrollo utilizarán el generador de analizadores lexicográficos, *Alex*. Posteriores entregas requerirán de *Happy* para el análisis sintáctico.

Análisis Teórico-Práctico

A continuación se presentan una serie de preguntas que deberá responder en un archivo de texto plano cuyas especificaciones serán detalladas más adelante.

1. Proporcione una expresión regular E que corresponda a los comentarios definidos en `Lanscii`. Dé el diagrama de transición (la representación gráfica) de un autómata finito (posiblemente no-determinístico) M que reconozca el lenguaje L denotado por E. De acuerdo con la definición de los comentarios en Lanscii, explique brevemente **cuáles** son las decisiones tomadas al construir la expresión regular E para que efectivamente sean cumplidas las restricciones de un comentario válido y **porqué** éstas funcionan.

2. Proporcione dos expresiones regulares: *E0* y *E1* para el reconocimiento de la palabra reservada `write` y los identificadores de variables del lenguaje respectivamente.

3. Dé los diagramas de transición (la representación gráfica) de dos autómantas finitos (posiblemente no-determinísticos): *M0* y *M1* que reconozcan los lenguajes *L0* y *L1* denotados por *E0* y *E1* respectivamente.

4. Proponga el diagrama de transición de un autómata finito no-determinístico *M2* que reconozca la unión de los lenguajes *L(M0)* y *L(M1)*.

5. Un analizador lexicográfico debe ser capaz de discernir a cuál lenguaje pertenece una palabra (o, en este caso, *token*) que acaba de reconocer. De acuerdo con esto, cada estado final del autómata *M2* debe ser capaz de diferenciar si la palabra reconocida pertenece al lenguaje *L(M0)* o *L(M1)*. Inidique lo anterior en cada estado final de *M2*.

6. La asignación anterior de estados finales a lenguajes debe de crear conflictos de reconocimiento, indique cuáles son estos problemas y porqué ocurren.

7. De acuerdo con la pregunta anterior, indique cuáles son los conflictos del autómata propuesto *M2*, especificando las palabras que los generan, los lenguajes y estados finales involucrados.

8. Diga cuál solución puede ser utilizada para resolver los conflictos de reconocimiento desarrollados en las preguntas 6 y 7. Explique brevemente su solución y por qué funciona.

9. ¿Cómo relaciona Ud. el desarrollo de las preguntas 2-8 con la implementación de su analizador lexicográfico para el lenguaje `Lanscii`?

Entrega

Formato de Entrega

Deben enviar un correo electrónico a **todos los preparadores** con el asunto: `[CI3725]eXgY` donde `X` corresponde al número de la entrega e `Y` al número del equipo. El correo debe incluir lo siguiente:

Un archivo **.zip** con el nombre `eXgY` siguiendo las mismas instrucciones del asunto del correo referentes a los valores de `X` e `Y`, que contenga:

- Código fuente debidamente documentado.
- En caso de utilizar *Haskell*, deben incluir un archivo Makefile o un archivo de configuración para *Cabal*. En caso de utilizar *C++* se debe incluir un archivo Makefile. Si su proyecto no compila, el proyecto no será corregido.
- Un archivo de texto con el nombre `LEEME.txt` donde **brevemente** se expliquen:
  - Decisiones de implementación
  - Estado actual del proyecto
  - Problemas presentes
  - Cualquier comentario respecto al proyecto que consideren necesario
  - Este archivo debe estar identificado con los nombres, apellidos y carné de cada miembro del equipo de trabajo.
- Un archivo de texto con el nombre `analisistp.txt` con la respuesta a cada una de las preguntas correspondientes a la sección de **Análisis Teórico-Práctico**. En caso de ser necesario adjuntar imágenes para contestar alguna de las preguntas, éstas deben estar en formato **.jpg** o **.png** debidamente identificadas tanto en el archivo **.zip** como en el archivo `analisisTP.txt`.

Fecha de entrega

La fecha límite de entrega del proyecto es el día **domingo 03** de mayo de 2015 (semana 4) *hasta* las **11:50pm**, entregas hechas más tarde tendrán una

penalización del 20% de la nota, esta penalización aplica por cada día de retraso.