Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3725 - Traductores e Interpretadores

Bitiondo - Etapa I

Análisis Lexicográfico (5%)

Especificación de la entrega

En la primera etapa de desarrollo del interpretador para el lenguaje *Bitiondo* se debe implementar su analizador lexicográfico. 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.

Deben de considerar los siguientes como *tokens* de *Bitiondo*:

- Las palabras reservadas del lenguaje *Bitiondo*.
- Los identificadores de cada variable. El *token* para todos los identificadores tendría que ser el mismo (ver ejemplo).
- Los valores constantes permitidos por el lenguaje *Bitiondo*:
  - Los números enteros, cuyo contenido del *token* asociado sea el número reconocido.
  - Las cadenas de caracteres encerradas entre comillas dobles, cuyo *token* asociado tendrá como contenido la cadena en cuestión.
  - Las constantes booleanas.
  - Las expresiones de tipo `bits`.
- Los símbolos utilizados para operadores y separadores.

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 `bitiondo` y recibirá como primer argumento el nombre del archivo con el programa en *Bitiondo* a analizar.

Por salida, 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 programa en *Bitiondo* posee errores léxicos, se debe mostrar por salida **todos** los errores encontrados y **sólo** los errores encontrados. Es decir, la salida **no** puede contener tanto *tokens* como errores léxicos del programa. Asimismo, deben recordar que los errores léxicos no son considerados como *tokens* reconocidos por el lenguaje.

Ejemplo de un programa correcto en *Bitiondo* y su salida de análisis lexicográfico respectiva:

```
begin
  int x;
  int y;

  input x;
  outputln "Hola, soy la variable x, valgo: ", x;
  y = x + 2;
  # Soy un comentario y no un token.
end
```

Salida:

```
$ ./bitiondo program.bto
begin at line 1, column 1
int at line 2, column 3
identifier at line 2, column 7 with value `x`
semicolon at line 2, column 8
int at line 3, column 3
identifier at line 3, column 7 with value `y`
semicolon at line 3, column 8
input at line 5, column 3
identifier at line 5, column 9 with value `x`
semicolon at line 5, column 10
outputln at line 6, column 3
string at line 6, column 12 with value `"Hola, soy la variable x, valgo: "`
comma at line 6, column 46
identifier at line 6, column 48 with value `x`
semicolon at line 6, column 49
identifier at line 7, column 3 with value `y`
assign at line 7, column 5
identifier at line 7, column 7 with value `x`
plus at line 7, column 9
integer at line 7, column 11 with value `2`
semicolon at line 7, column 12
end at line 9, column 1
```

Y, por otro lado, un ejemplo de programa incorrecto con su salida asociada:

```
begin {
  # Otro comentario
```

```
  int x;;
  input y;
}
```

Salida:

```
Error: Se encontró un caracter inesperado "{" en la Línea 1, Columna 7.
Error: Se encontró un caracter inesperado "}" en la Línea 5, Columna 1.
```

Noten que para la salida de un programa *correcto* en *Bitiondo*, en esta primera etapa de desarrollo, se imprime el toda la informaición interesante del *token*, como cuál fue el econtrado, su posición y su información asociada. 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 *Bitiondo*, pueden escoger uno (1) de los tres (3) lenguajes de programación a continuación. Para cada uno de ellos se indica las herramientas disponibles para el desarrollo de un interpretador de código. La versión escogida para cada uno es la disponible en los repositorios de Ubuntu LTS 16.0.4 (*Xenial Xerus*). Si desea utilizar otro sistema operativo u otras versiones, debe garantizar que su código funcione para estas versiones.

- *Ruby:*
  - *ruby* 2.3.0
  - Para esta etapa de desarrollo no se utilizará una herramienta en *Ruby* que permita el análisis lexicográfico, por lo tanto, el trabajo para esta entrega se debe realizar a través del manejo de las expresiones regulares del lenguaje. Para entregas posteriores se utilizará *Racc* para el análisis sintáctico.
- *C++:*
  - Usando *GCC* 5.3.1
  - Para esta etapa de desarrollo se usará *Flex* 2.6.0. Posteriores etapas requerirán de *Bison* para el análisis sintáctico.
- *Haskell:*
  - *GHC* 7.10.3
  - *Alex* y *Happy*. Para esta etapa de desarrollo utilizarán el generador de analizadores lexicográficos, *Alex*. Posteriores entregas requerirán de *Happy* para el análisis sintáctico.

Entrega

Metodología de trabajo y entregas

El proyecto en cada una de sus etapas será realizado en parejas. Cada pareja deberá crear un repositorio **privado** en *GitLab* y agregar a Matteo Ferrando (@chamini2) y a Melecio Ponte (@melecio) bajo la figura de *Guest*. El repositorio debe llevar por nombre `CI3725-Bitiondo`.

El repositorio debe contener:

- Código fuente debidamente documentado.
- En caso de utilizar Haskell o C++, deben incluir un archivo Makefile o un archivo de configuración para *Cabal*. **Si su proyecto no compila, el proyecto no será corregido**.
- Un archivo de texto con el nombre `README.md` donde **brevemente** se explique:
  - 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.

Para la fecha de la entrega, se revisará el último *commit* de fecha y hora previas a la fecha y hora límite, con la etiqueta `v0.1.x` donde `x` es de libre uso por los integrantes del equipo para sus versiones internas. Usaremos estas etiquetas durante cada una de las etapas, de manera que `v0.2.x` corresponderá a la 2da etapa y así sucesivamente.

En caso de que no existir etiquetas, se usará el último *commit* de fecha y hora previas a la fecha y hora límite.

Fecha de entrega

La fecha límite de entrega del proyecto es el día **viernes 20** de octubre de 2017 (semana 5) *hasta* las **11:50pm**, entregas concretadas más tarde tendrán una **penalización del 20%** de la nota, esta penalización aplica por cada día de retraso. En caso de que deseen que se revise un *commit* que incurra en penalizaciones, deben hacerlo saber por correo a cualquiera de los encargados de la materia.