Escuela de Ingeniería en Computación

Scanner/Analizador léxico para el lenguaje de alto nivel *ABC.*

Estudiantes:

Roberto Rojas Segnini, 2016139072

Daniel Alvarado Bonilla, 2014089192

Profesora Erika Marín

Compiladores e Intérpretes

Grupo 2

II Semestre del 2018

TEC | Tecnológico de Costa Rica

Introducción

Suele decirse que *un viaje de mil kilómetros empieza con un solo paso*. En el mundo de los compiladores, ese primer paso es el *scanner*; también conocido como *analizador léxico* o *tokenizador*. Este primer paso permite separar y organizar el código fuente mediante la lectura y categorización de los caracteres.

El *scanner* se encarga de leer el código fuente para encontrar aquellos grupos de caracteres que pertenecen al lenguaje en cuestión y aquellos que no. Todos los lenguajes poseen reglas diferentes que determinan si una palabra es válida o no.

Además, el *scanner* se encarga de categorizar los tokens debidamente validados y aceptados. Los tipos de tokens dependen de cada lenguaje y de sus necesidades. Generalmente, los tokens se separan en **literales** (números, *strings*, caracteres explícitamente tipificados), **identificadores** (nombres de variables, clases, objetos, etc.), **operadores** (símbolos utilizados para especificar diversos tipos de operaciones) y las **palabras reservadas** (palabras propias del lenguaje).

Para poder determinar si una tira de caracteres pertenece o no a un lenguaje, el *scanner* utiliza **expresiones regulares**, las cuales especifican las características de los tipos de tokens existentes. Estos tipos, a su vez, determinan si una tira para pertenecer al lenguaje.

En este documento, se presenta la implementación de un *scanner* para un lenguaje llamado ABC. Este es lenguaje de alto nivel, insensible a las mayúsculas y minúsculas (*case insensitive*), el cual posee identificadores; operadores lógicos, aritméticos, de orden y

de condición, palabras reservadas y literales del tipos *string,* enteros, flotantes, caracteres y flotantes escritos en notación científica.

Estrategia de solución

Para crear el compilador, se utilizó la librería para generar analizadores léxicos JFlex, escrita en Java. Esta herramienta recibe las distintas expresiones regulares que conforman un lenguaje, así como sus acciones correspondientes, para generar un programa que lee y analiza los caracteres presentes en algún documento especificado. De este modo, se determina su pertenencia al lenguaje y el tipo del token (JFlex, 2018).

Las expresiones regulares fueron separadas en dos grupos: **tokens** y **errores**. Las expresiones regulares para los tokens contienen las especificaciones de pertenencia al lenguaje (por ejemplo, los identificadores no pueden empezar con número, los enteros deben escribirse de la forma X.0, etc.). En caso de no cumplir con alguna de las reglas, las expresiones regulares para los errores se encargan de catalogar el problema presentado (error de identificador, de literal numérica, punto flotante, *string*, entre otros).

En cuanto a las acciones de cada expresión regular, estas deben ser acomodadas estratégicamente para evitar errores de clasificación errónea. En general, el *scanner* primero busca entre algunas expresiones de error por posibles fallos en la escritura del código fuente. Una vez superada esta etapa, el analizador utiliza las expresiones para los literales, operadores, identificadores y palabras reservada. Si al finalizar todas las expresiones regulares no se obtiene ningún *match*, el *scanner* registra ese conjunto de caracteres como un error léxico de caracteres inválidos.

Análisis de resultados

Id	Actividad	Descripción	Porcentaje de realización	Justificación
1	Leer archivo	Leer correctamente el archivo de texto que posee en código fuente	100%	
2	Listado de errores léxicos	Se muestran todos los errores léxicos encontrados.	90%	Existencia de errores en el análisis.
3	Listado de tokens encontrados	Se despliegan todos los tokens válidos dentro del lenguaje	90%	Existencia de errores.
4	Tipos de tokens	Los tokens encontrados son categorizados de manera correcta.	90%	Existencia de errores.
5	Líneas y ocurrencias	El listado de tokens presenta de manera correcta la cantidad de apariciones de un token, así como la línea donde se encuentra.	100%	

Lecciones aprendidas

1. Si se desea categorizar los errores, se deben crear expresiones regulares que especifiquen el conjunto de caracteres que generarían dicho error.

2. Entre mayor sea el lenguaje, más complejo se vuelve el manejo de las expresiones regulares y sus casos especiales.

Casos de pruebas

Para literales

Para determinar que el análisis y categorización de los literales se realiza correctamente, se utilizó el siguiente código fuente:

```
"Este es un String"

#3  #45 #123 #765  //estos son literales de caracteres

" un string de
varias
lineas"

"ljhgsdkjaghsdf ;  //String sin cerrar

{Ahora vienen numeros enteros}
45
89

123
324
+304  //separar el operdor del numero

-989  //separar el operdor del numero

{Ahora vienen numeros flotantes}

1.23
123.345
```

```
123.  // error porque no tiene nada despues del punto

123.22
.22  //error.
.67 // No tiene nada antes del punto

+1.23   //separar el operdor del numero

-123.345 //separar el operdor del numero

{Con exponentes}

3.0E5
1.5E-4
-123E10  //error el numero antes del exponente debe ser real
123.5E  //error no tiene nada despues del exponente

5.4E3.4 // error el numero despes del e tiene que ser entero
```

Resultados obtenidos

El *scanner* analiza correctamente, en su mayoría, las tiras de caracteres ingresadas y determina de manera notable la pertenencia de estas al lenguaje. Sin embargo, la tipificación de los errores y tipos no es del todo satisfactoria: por ejemplo, la línea `5.4E3.4` debería mostrar un ERROR_FLOATING_POINT, puesto que el valor del exponente no puede ser un número en punto flotante (en este caso, 3.4). Pero el programa toma el valor "**5.4E3** como un número en notación científica y, de forma separada, el valor **.4** como un ERROR_FLOATING_POINT porque es necesario que el valor sea escrito de la forma **0.4**, con el cero antes del punto.

Para identificadores

Para determinar que el análisis y categorización de los identificadores se realiza de manera satisfactoria, se utilizó el siguiente código fuente:

```
X x

hola
x Y
 Y
S
S X
S
Hola
X y

s s S

HoLa holA
```

Resultados obtenidos

El análisis de los identificadores se realizó de manera correcta: los identificadores son separados correctamente, se clasifican como IDENTIFIER.

Para operadores

Para determinar que el análisis y categorización de los operadores se realiza de manera satisfactoria, se utilizó el siguiente código fuente:

```
, ;


(  )  [  ]
:=  .   :   +=  -=  *= /=
>> << <<= >>=
```

Resultados obtenidos

Todos los operadores son tokenizados correctamente.

Para palabras reservadas

```
{Con este ejemplo se quiere probar los diferentes tipos de comentarios y
ademas que reconozca tanto las palabras reservadas como los operadores}

AND ARRAY
BEGIN BOOLEAN BYTE
CASE CHAR CONST
DIV DO DOWNTO
ELSE END
FALSE FILE FOR FORWARD FUNCTION
GOTO
IF IN INLINE INT
LABEL LONGINT
MOD
NIL NOT
OF OR
PACKED PROCEDURE PROGRAM
READ REAL RECORD REPEAT
SET SHORTINT STRING
THEN TO TRUE TYPE
```

```
UNTIL
VAR
WHILE WITH WRITE XOR

(* POr ultimo este es el otro tipo de comentarios
Tambien es de Boloque *)
```

Resultados obtenidos

Todas las palabras reservadas se analizan y categorizan correctamente.

Bitácora de trabajo

Leyendas de los integrantes:

Roberto Rojas Segnini

Daniel Alvarado Bonilla

Fecha	Descripción de la tarea
Miércoles 19/09/2018	Instalación de JFlex.
Jueves 20/09/2018	Establecimiento de tipos de tokens (enumerators). Creación de plantilla de JFlex. Definición de expresiones regulares básicas.
Viernes 21/09/2018	Definición avanzada de expresiones regulares. Definición y gestión de errores.
Sábado 22/09/2018	Pruebas y control de *bugs*. Creación de listados de tokens y errores.

Bibliografía

Búcaro, J. (2015). *Análisis Léxico usando JFlex*. Recuperado de: https://jonathanbucaro.com/2015/04/26/analisis-lexico-usando-jflex/

JFlex (2018). *JFlex User's Manual*. Recuperado de: http://jflex.de/manual.html

Manpagez (2012). *6 Patterns*. Recuperado de: http://www.manpagez.com/info/flex/flex-2.5.37/flex_10.php#Patterns