Memoria Práctica 1 de Laboratorio de Procesadores del Lenguaje: Analizador Léxico

Cristina Corral Muñoz

20 de Octubre de 2015

1 Introducción

En esta práctica se implementará un analizador léxico usando JFLex para textos con una sintaxis parecida a los lenguajes de programación estructurados.

2 Tratamiento de los tipos de tokens

2.1 Palabras reservadas

Las palabras reservadas se ponen de forma literal como macros individuales: de este modo se declararía PROGRAM = "PROGRAM" y de igual manera con todas las reservadas.

Por degracia, %ignorecase no respondía correctamente y no conseguía evitar distinciones entre mayúsculas y minúsculas así que se cambió la definición a PROGRAM = [Pp][Rr][Oo][Gg][Rr][Aa][Mm].

2.2 Identificadores y Funciones

Los identificadores tienen como regla empezar siempre por una letra del alfabeto y contener después una serie de números y letras, estando bien 'num1' pero no '1num'.

Como, en un principio, en el ejemplo anterior el análisis devolvía la existencia de un integer (el 1), y luego de un identificador (num), se ha añadido un estado llamado estadoToken que sólo permitirá la existencia de espacios, signos de puntuación, espacios, símbolos de operaciones matemáticas y saltos de línea una vez haya identificado un token correcto.

Nota: Una palabra reservada mal escrita (con símbolos aceptados en identificadores) se considerará un identificador puesto que se puede tener en cuenta que el escritor en cualquier momento puede desear tener un identificador llamado "programa" o "prog1" y en ningún momento quiso referenciar la palabra reservada "program".

2.3 Integers y Booleanos

Los tipos más simples de token son los booleanos puesto que sólo son true o false tanto en minúsculas como en mayúsculas.

Los integer pueden llevar antes del número un símbolo "+" o "-" dependiendo de si el número es positivo o negativo.

Tanto booleanos como integers están prohibidos durante la ejecución del estado estadoToken y serán detectados como error léxico.

2.4 Comentarios

Los comentarios están gestionados por un estado especial llamado estadoComentario que admitirá cualquier elemento una vez encuentre dos barras "//" y seguirá ignorando los elementos hasta encontrar un salto de línea. Esto también puede gestionarse con la regex = "//".*\n.

2.5 Errores

La gestión de errores se realiza con otro estado llamado estadoError. Este tiene como función recorrer todo el token hasta que encuentre un espacio, un punto y coma, una coma o un salto de línea.

En el siguiente punto se explicará con más detalle cómo se entra en un estado de posible error, se detecta y se sugiere una corrección.

2.5.1 Gestión de los Errores

En un principio podemos tener dos casos: que se detecte un elemento ajeno no descrito en el regex o que se detecte un elemento seguido (sin espacios) de uno no descrito. En el primer caso se pasa al estadoError directamente y se dispone por pantalla un mensaje de error junto con la palabra errónea (guardada en la clase como atributo String llamado "error"), y se muestra una sugerencia. Dicha sugerencia se selecciona mediante un sencillo algoritmo: dentro de un diccionario de palabras correctas (llamado diccionario.dic) se busca con el método .contains() palabras que coincidan con nuestro error; la palabra que tenga mayor número de coincidencias será la sugerida.

En el caso de encontrarnos con un token correcto seguido de uno incorrecto (por ejemplo: "identificador??") para que el compilador no tenga como buena "identificador" y pase como mala "??" se ha metido el estado ya mencionado anteriormente como estadoToken. En el atributo de clase "key" se guardará temporalmente el token correcto, por si resulta ser incorrecto, poder quitarlo de la tabla de símbolos (nota: en la tabla de símbolos solo se añaden los nombres de identificadores y de funciones) que internamente tiene la estrutura de un Hash.

3 Tabla de símbolos

La tabla de símbolos es similar a la que hice (que es similar a su vez de la proporcionada en el enunciado de la práctica) el año anterior.

Su funcionamiento es bastante básico: cuando se detecta un token correcto que se desee guardar (todos menos símbolos, espacios, puntuación y parecidos), se incrementa un contador. En el caso de las funciones y de los identificadores también se ha incorporado una tabla Hash para guardar los nombres sin que se repitan.

4 Ejemplo de entrada y salida

Los ficheros de entrada proporcionados para esta práctica se llaman "test" seguido de un número entre 1 y n y terminados en ".prog". La entrada se hace vía argumento.

A continuación se muestra un ejemplo del programa dado en la práctica con varios cambios para que arroje o no ciertos errores:

```
PROGRAM
VArDECL
        id,id2,id3 -> INTEGER;
        id4,id5,id6 -> BOOLEAN;
        cont, 1asd -> integer;
END;

function $nombre1 a1 (boolean), 1b (integer)
        id3 = 4 + b1;
        // Comentario dentro de funcion
        return true;

function $nombre2 a2
        return 4;

function $nombre3
        return true;

BeGIn
        //fichero de prueba PROGRAM
        id3 = (6 + 6) - $nombre2(2);
        id=$nombre2(4);
        id4= $nombre1(false,2);
        id7=$nombre2(id);
END;
```

Devolverá el analizador al ser ejecutado:

->Error lexico en la linea 5 en el elemento: 1asd quizá quiso decir: asd
->Error lexico en la linea 8 en el elemento: 1b quizá quiso decir: b1
Identificadores: [a2, id, a1, id7, id6, id5, b1, id4, id3, id2, cont]
Número: 16
Palabras reservadas: [PROGRAM, INTEGER, BeGIn, VArDECL, BOOLEAN, function, integer,
return, boolean]
Numero de palabras reservadas: 14
Numero de integers: 7
Numero de booleanos: 3
Funciones: [$nombre3, $nombre2, $nombre1]
Numero de funciones: 3
Numero de errores .....: 2