

**Procesadores de Lenguajes**

**Helena García González,**

**Elena Mateos Martín,**

**Miguel Ángel Monreal Velasco**

Memoria

*2017*

Índice

[Introducción al problema 3](#_Toc479331789)

[Desarrollo de la solución al léxico 4](#_Toc479331790)

[Desarrollo de la solución al sintáctico 5](#_Toc479331791)

[Parte obligatoria 5](#_Toc479331792)

# Introducción al problema

En la práctica, se pide realizar un procesador de lenguajes para un subconjunto de un lenguaje cercano a C. Para cumplir con este cometido, se utilizarán *jflex*  y *cup.*

# Desarrollo de la solución al léxico

Hemos definido diez macros para resolver este problema y así simplificar el uso de las expresiones regulares que determinan. Además, hemos declarado dos estados exclusivos para el reconocimiento de constantes literales y comentarios de más de una línea.

El estado inicial YYINITIAL está compuesto por ocho patrones. Los tres primeros son las macros que se relacionan con el reconocimiento de los identificadores, constantes enteras y decimales. Les siguen los patrones para reconocer las constantes literales y los comentarios tanto de una línea como de varias. Por último, para el tratamiento de errores, es decir si el lexema no se corresponde con ninguno de los patrones anteriores, se imprimirá dicho error con la cadena no aceptada. A su vez, si en la entrada se encuentra con un salto de línea, lo imprime.

El estado ENTRECOMILLADO se utiliza para aceptar las constantes literales. Para llegar a este estado desde el inicial, se ha de transitar con una comilla. Después, en ENTRECOMILLADO, se genera un bucle hasta que se encuentra una comilla, entonces vuelve al estado inicial y así seguir procesando el resto de la entrada.

El estado COMENTARIO permite reconocer comentarios de más de una línea. Se comienza desde el estado inicial y se transita con “/\*” a COMENTARIO en el cual se queda en ese mismo estado con cualquier símbolo hasta que encuentre “\*/” y en ese momento retorna al estado inicial para examinar el resto de la entrada.

# Desarrollo de la solución al sintáctico

## Parte obligatoria

Para esta sección hemos utilizado la clase *java* generada a partir del archivo *flex* generado por *Jflex* y *Cup*  para transformar nuestra gramática en .*cup* a dos clases *java*, *parser*  y *sym*.

Primero, se pasó la gramática impuesta en el ejercicio a un archivo cup, enlazando el nombre de los terminales de la especificación léxica con los terminales de la gramática e incluyendo los no terminales de las reglas de la misma. Aquí añadimos un operador denominado *precedence*, que sirve para definir la predominancia de los operadores multiplicación, división y módulo sobre la suma y la resta. Esta incorporación es necesaria, debida a la ambigüedad ante la que se puede encontrar el analizador sintáctico con cadenas que contengan, por ejemplo, 3 + 4 \* 5, ya que debe reducir primero 4\*5, en vez de 3 + 4.

Después generamos las clases *parser* y *sym* mediante *Cup* por consola, para crear un proyecto java con las clases mencionadas junto con la generada por *Jflex* y la principal para poder ejecutar con un fichero de entrada.

## Parte optativa

En la realización de la parte optativa simplemente añadimos las reglas nuevas y otra sentencia *precedence* para las operaciones *and* y *or*.

Finalmente, generamos las clases con *Cup* e incluimos nuevas líneas en el *flex* para poder generar tokens necesarios en esta parte.

# Casos de prueba

## Casos correctos

### Caso 1

int calcular(int var5, int var6){

int var7;

var7=var5+var6;

}

### Caso 2

float restar(float t){

int $minuendo;

int $sustraendo;

int $diferencia;

$diferencia = $minuendo - $sustraendo;

return $diferencia;

}

Ambos ejemplos son válidos, el analizador n encuentra ningún error.

## Casos erróneos

### Caso 3

void calcular(int i5, int i6){

int i8

i7=i5+i6;

}

Este fragmento de código tiene el típico error del punto y coma, por lo que el analizador sintáctico encuentra un error al intentar procesar la cadena de entrada.

### Caso 4

floar metodo(){

float f = 35.5;

return f;

}

Aquí se genera un error al principio de la cadena, ya que no está reflejado que haya un tipo llamado floar.

# Traducción dirigida por sintaxis

Tras haber implementado el léxico y el sintáctico pasamos a la última etapa que es desarrollar un traductor dirigido por sintaxis. Al igual que en las partes anteriores, teníamos dos partes:

## Parte obligatoria

El objetivo era poder visualizar un programa implementado con el lenguaje C en html con un formato específico. Dicho archivo html se llamaría igual el fichero de entrada introducido para generar el traductor e internamente estaría estructurado con un title con el mismo nombre y un encabezado que mostraría “Programa” seguido del nombre del fichero.

A continuación aparecería el apartado de las cabeceras de funciones introducido por un encabezado de segundo nivel “Funciones” y que mostraría una lista con las cabeceras de los métodos que al pulsar sobre ellas nos llevasen a la función correspondiente.

Para finalizar estaría el apartado de las funciones que estaría separadas por una línea y al final de ellas habría otros dos enlaces (uno al inicio de la página web y otro al inicio de la función). Se debía mostrar la cabecera de la función junto con el cuerpo de esta; dicho cuerpo estaría dentro de un div que le daría un estilo determinado y entre etiquetas de html que le darían un formato de código a las sentencias.

Además teníamos que darle un estilo concreto a las constantes, palabras reservadas e identificadores mediante un span.

## Parte opcional

En esta parte deberíamos adaptar la parte optativa del sintáctico siguiendo las mismas reglas anteriores. Había que incluir bucles y estructuras pero ambas seguían el mismo formato. Las estructuras deberían estar al final de la página web mientras que los bucles en el trozo de código que le correspondía.

## Desarrollo de la solución

Para desarrollar la solución con las siguientes especificaciones se emplearía el archivo cup generado en el punto anterior incluyendo el código java necesario para que tenga el funcionamiento deseado.

Se incluirían acciones al final o entre las reglas que me cogiese los valores de los no terminales o terminales que necesitamos (otorgándoles un identificador único para acceder a ellos en el código) y desarrollase el código.

Se usaron atributos heredados todo el tiempo y empleamos listas o strings para guardar los valores deseados e imprimirlos después de procesar todo el fichero de entrada en un fichero html.

# Bibliografía

Manual de CUP:

<http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html>