# Proyecto de Autómatas y Lenguajes

*Curso 2014-2015*

## Práctica 2: ANÁLISIS MORFOLÓGICO

**Fecha de entrega: Semana del 3 de noviembre, antes del inicio de la clase.**

## Objetivo de la práctica:

El objetivo de la práctica es la programación del analizador morfológico del compilador utilizando la herramienta Flex. El analizador morfológico (analizador léxico o scanner) es la parte del compilador que se encarga de reconocer las unidades sintácticas (*tokens*) del programa. Junto con el analizador morfológico se entregará un programa escrito en C para probarlo.

## Desarrollo de la práctica:

### 1. Codificación de una especificación para Flex.

Partiendo de la gramática del lenguaje Alfa, se identificarán las unidades sintácticas (*tokens*) y se diseñará el conjunto de expresiones regulares que las representan. A partir de estas expresiones regulares se codificará en el fichero ***alfa.l*** la especificación correspondiente para la herramienta Flex. Es importante prestar especial atención al orden en el que se colocan las reglas para la detección de los tokens para evitar que unos patrones oculten a otros.

Junto a las reglas necesarias para la gestión de las unidades sintácticas (*tokens*) del lenguaje, se añadirán, si es necesario, reglas para:

- Ignorar los espacios, tabuladores y saltos de línea.
- Ignorar los comentarios (recuérdese que en Alfa un comentario comienza con // y acaba al final de la línea).
- Gestionar los errores morfológicos (sólo se considerarán errores los símbolos no permitidos y los identificadores que excedan de la longitud máxima).

La acción asociada a cada regla será un *return* de un valor numérico que identifique al tipo de patrón identificado. Los valores numéricos asociados a cada tipo de token están definidos en el fichero *tokens.h* facilitado a través de la plataforma moodle.

### 2. Codificación del programa de prueba.

Una vez construida la especificación Flex para el compilador y generado el programa *lex.yy.c* (que incluye la función *yylex()*), se desarrollará un programa en C para probar el analizador morfológico. Dicho programa recibirá como argumentos los nombres de dos ficheros. Utilizará el primer fichero como entrada, invocará a la función *yylex()* para realizar el análisis morfológico del mismo, y escribirá los resultados del análisis en el segundo fichero. A continuación se describen los ficheros de entrada y salida, y se muestran algunos ejemplos.

**3. Descripción del fichero de entrada.**

El fichero de entrada contiene texto plano correspondiente a un programa escrito en el lenguaje Alfa (no necesariamente correcto).

**4. Funcionalidad del programa.**

Haciendo uso del analizador morfológico construido con Flex, el programa de prueba deberá detectar en el fichero de entrada los siguientes patrones:

- Palabras reservadas.
- Símbolos.
- Identificadores (téngase en cuenta las reglas para la construcción de identificadores especificadas en la descripción de la gramática).
- Constantes (enteras o booleanas).
- Errores (símbolos no permitidos e identificadores que excedan de la longitud máxima).

Cada vez que se detecte un token válido, el programa de prueba informará de ello, indicando el tipo de token detectado. Cada vez que se detecte un error, el programa avisará del mismo indicando el tipo de error así como la línea y la columna en la que aparece (se considerará que la primera fila del fichero es la número 1, y la primera columna de cada línea es la número 1). Es necesario por tanto llevar la cuenta del número de líneas (que se incrementará cada vez que aparezca un salto de línea) y el número de columnas (que se incrementará tras la detección de un token, un espacio, un tabulador, un comentario o un símbolo no permitido, y se inicializará cada vez que aparezca un salto de línea). Para incrementar correctamente el número de columnas puede usarse la variable *yyleng*, que almacena la longitud de cada token detectado.

**5. Descripción del fichero de salida.**

El fichero de salida tendrá una línea por cada uno de los tokens detectados en el fichero de entrada. En cada línea se mostrará:

- El tipo de token detectado. Se utilizará el nombre usado en el fichero *tokens.h*.
- El código numérico del token (definido en el fichero *tokens.h*).
- El lexema analizado (valor de la variable *yytext*).

Por ejemplo, si en el fichero de entrada nos encontramos con la palabra *printf*, en el fichero de salida se escribirá:

```
TOK_PRINTF          109    printf
```

En caso de que aparezca algún error, se informará del mismo mostrando el tipo de error, la fila y la columna del fichero en el que aparece (véanse los ejemplos facilitados en moodle para ver el formato a seguir para informar de los errores).

**Nota: es muy importante respetar el formato de los ficheros de salida puesto que la corrección se realizará de manera automática.**

**6. Ejemplos.**

Para probar inicialmente el código desarrollado se facilitan dos ficheros de entrada (*entrada1.txt* y *entrada2.txt*) con sus salidas correspondientes (*salida1.txt* y *salida2.txt*). Suponiendo que el programa ejecutable se denomina *pruebaMorfo*, la siguiente instrucción:

```
pruebaMorfo entrada1.txt misalida1.txt
```

debe generar un fichero con nombre *misalida1.txt* que sea idéntico al fichero *salida1.txt*. En particular, al hacer:

```
diff -bB salida1.txt misalida1.txt
```

no debe encontrarse ninguna diferencia entre los dos ficheros.

## Entrega de la práctica:

Se entregará a través de Moodle un único fichero comprimido (*.zip*) que deberá cumplir los siguientes requisitos:

- Deberá contener todos los fuentes (ficheros *.l*, *.h* y *.c*) necesarios para resolver el enunciado propuesto. No es necesario incluir el fichero *lex.yy.c* puesto que puede generarse a partir del *.l*.
- Deberá contener un fichero *Makefile* compatible con la herramienta make que para el objetivo ***all*** genere el ejecutable de nombre ***pruebaMorfo***.
- El nombre del fichero *.zip* será ***Apellido1_Apellido2_Nombre_morfo.zip.***

Se recuerda al alumno que **las prácticas son incrementales por lo que una práctica aprobada pudiera conllevar errores importantes en las siguientes fases. Es responsabilidad del alumno subsanar totalmente los errores en cada fase, dando correcto cumplimiento a los requerimientos de los enunciados.**