Un analizador sintáctico de expresiones regulares

En esta práctica se aplicará el método de análisis sintáctico descendente recursivo para reconocer expresiones regulares. La gramática de las expresiones regulares que se utilizará es la que se muestra en la Figura 4.1.

Expresión → Término Resto_expresión
Resto_expresión → | Expresión
Resto_expresión → ε
Término → Cotermino Resto_término
Resto_término → Término
Resto_término → ε
Cotérmino → Factor Resto_cotérmino
Resto_cotérmino → * Resto_cotérmino
Resto_cotérmino → ε
Factor → (Expresión)
Factor → id
Factor → &

Figura 4.1 Una gramática para expresiones regulares

En la gramática, el símbolo **&** denota la expresión regular vacía y los identificadores (**id**) están restringidos a una sola letra.

Se leerá un fichero que contiene en cada línea una expresión regular y el programa imprimirá en pantalla para cada línea si se reconoce o no la expresión regular (se reconocerá si está bien construida según la gramática anterior). En caso de que la expresión no sea correcta, se imprimirá la posición en la cadena de entrada del símbolo que ha producido el error.

Se recomienda diseñar mediante un tipo enumerado el conjunto de tokens (símbolos terminales) posibles.

El alumno deberá diseñar (como mínimo) las siguientes funciones:

- Una función *get_token()* que devuelva al analizador sintáctico el siguiente token (componente léxico) que encuentre en la cadena de entrada. Esta función es la que realizará el análisis léxico de la entrada. La función convertirá cadenas de caracteres en tokens, de forma que el analizador sintáctico trabajará con tokens, no con caracteres.
- Una función para cada símbolo no terminal de la gramática.
- Una función *match()* que elimine de la entrada un token esperado, o bien genere un error si el token que encuentra en la entrada no es el esperado. A esta función se le pasará como parámetro el token que se espera encontrar.

El análisis sintáctico de una expresión comienza con la llamada a *get_token()* para conseguir el símbolo de preanálisis (lookahead) y una llamada a la función correspondiente al no terminal *Expresión*. Esta función analizará la expresión, y a continuación se debe invocar la función *match()* para eliminar de la entrada un carácter de fin de línea ('\0').

Será conveniente disponer de una variable que indique si se ha producido un error. En caso de error habrá que notificar la posición que se estaba explorando en la cadena de entrada.

En esta práctica, los First y Follow de las construcciones se incluirán en el código a mano, si son necesarios (el programa no debe calcular los conjuntos First y Follow para las variables sintácticas).

En el servidor ftp del CSI se puede obtener el fichero

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_Asintactico/test.in

que contiene algunas expresiones regulares. También en el mismo directorio están disponibles los programas

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_Asintactico/asin

ftp://ftp.csi.ull.es/pub/asignas/AUTOMALF/p04_Asintactico/as.exe

que tienen el comportamiento deseado.
Para realizar la práctica se puede utilizar cualquiera de las librerías de funciones que se han utilizado anteriormente y como referencia recomendamos de nuevo la obra de Aho [Aho90].