

PRÁCTICA 1: ANALIZADOR LÉXICO

Objetivo de la Práctica

- Esta práctica tiene como objetivo la codificación de un analizador léxico mediante el uso del lenguaje de programación C y la herramienta de ayuda a la generación de analizadores léxicos **flex**.
- Este analizador será utilizado por el compilador del lenguaje **ALFA** que va a desarrollarse durante el curso.

Desarrollo de la Práctica

Estudio de la Gramática

- Inicialmente se tratará de identificar en la gramática completa del lenguaje, incluida al final de este enunciado, el conjunto de unidades sintácticas o *tokens* existentes.

Codificación de una Especificación para **flex**

- El alumno deberá diseñar el conjunto de expresiones regulares que representa cada unidad sintáctica.
- **Observación muy importante:** Recuerde que algunas tareas que el analizador léxico puede y suele realizar son:
 - Ignorar los espacios en blanco y tabuladores, que actúan como separadores entre unidades sintácticas.
 - Ignorar los comentarios, que no aportan código ejecutable al programa.
 - Gestionar errores morfológicos.
- Se deberá escribir un fichero de nombre **alfa.l** que sea una especificación correcta para la entrada de la herramienta **flex**.
- Se recomienda al alumno que se familiarice primero con **flex** mediante las explicaciones que su profesor de prácticas realizará en su laboratorio y posteriormente diseñe el conjunto final de expresiones regulares.

Codificación de un Programa de Prueba

- Se deberá escribir (en C) un programa de prueba del analizador léxico generado con **flex**, con los siguientes **requisitos**:
 - Nombre del programa fuente: **prueba_lexico.c**
 - Al ejecutable correspondiente se le invocará de la siguiente manera:

prueba_lexico [<nombre_fichero_entrada> [<nombre_fichero_salida>]]

- Es decir, el programa se puede ejecutar:
 - Con 0 argumentos: se utilizan la entrada y la salida estándares.
 - Con 1 argumento: se utiliza la salida estándar y el argumento como nombre del fichero de entrada.
 - Con 2 argumentos: el primer argumento se interpreta como el nombre del fichero de entrada y el segundo como el nombre del fichero de salida.
- La estructura de los ficheros de entrada/salida se explican a continuación.

Descripción del Fichero de Entrada

- El fichero de entrada contiene un programa escrito en lenguaje *ALFA* (no necesariamente correcto).

Funcionalidad del Programa

- El programa de prueba, haciendo uso del analizador léxico construido con *flex*, deberá identificar en el fichero de entrada las siguientes unidades sintácticas:
 - Palabras reservadas: cualquier palabra reservada *ALFA* correcta.
 - Símbolos: cualquier símbolo *ALFA* correcto.
 - Identificadores: que sigan la sintaxis descrita en la gramática para el símbolo no terminal *<identificador>*.
 - Constantes numéricas: cualquier número entero o real.
 - Cualquier otra cosa, se considerará errónea.
- Por cada unidad sintáctica identificada en el fichero de entrada, el programa de prueba debe generar una línea en el fichero de salida según se describe a continuación.

Descripción del Fichero de Salida

- El fichero de salida se compone de un conjunto de líneas, una por cada unidad sintáctica del fichero de entrada. La estructura de la línea es la siguiente:
 - El primer elemento de la línea será, según corresponda
 - *TOK_PALABRA_RESERVADA*: para las palabras reservadas.
 - *TOK_SIMBOLO*: para los símbolos.
 - *TOK_IDENTIFICADOR*: para los identificadores.
 - *TOK_NUMERO*: para todos los números (enteros y reales).
 - *TOK_ERROR*: para los errores.
 - El segundo elemento será un número entero, que identificará la unidad sintáctica (debe ser distinto para cada unidad sintáctica). En la página web del laboratorio de Procesadores de Lenguaje está disponible un enlace para descargar el fichero **tokens.h** que contiene los valores numéricos de los tokens de la gramática de *ALFA*. Es obligatorio utilizar los valores de este fichero.
 - El tercer elemento será el *lexema* (el fragmento del programa fuente) analizado como la unidad sintáctica correspondiente.
 - **Muy importante: cada elemento de la línea estará separado por un tabulador y ningún otro carácter más**, aunque ofrezca un aspecto visualmente indistinguible.

Ejemplos

- Ejemplo 1: Si el fichero de entrada es el siguiente:

// Programa que eleva un número entero al cuadrado

```
main {
    int x, resultado;

    scanf x;
    resultado=x*x;
    printf resultado;
}
```

El fichero de salida es el siguiente:

```
TOK_PALABRA_RESERVADA  200  main
TOK_SIMBOLO  300  {
TOK_PALABRA_RESERVADA  201  int
TOK_IDENTIFICADOR  500  x
TOK_SIMBOLO  306  ,
TOK_IDENTIFICADOR  500  resultado
```

TOK_SIMBOLO	302	;	
TOK_PALABRA_RESERVADA	213	scanf	
TOK_IDENTIFICADOR	500	x	
TOK_SIMBOLO	302	;	
TOK_IDENTIFICADOR	500	resultado	
TOK_SIMBOLO	309	=	
TOK_IDENTIFICADOR	500	x	
TOK_SIMBOLO	303	*	
TOK_IDENTIFICADOR	500	x	
TOK_SIMBOLO	302	;	
TOK_PALABRA_RESERVADA	214	printf	
TOK_IDENTIFICADOR	500	resultado	
TOK_SIMBOLO	302	;	
TOK_SIMBOLO	301	}	

- Ejemplo 2: Si el fichero de entrada es el siguiente:

// Programa que eleva un número x a la potencia y

```
main {
    int x, y;
    int i, total;

    scanf x;
    scanf y;
    i=1;
    total=1;
    while (i<=y) {
        total=total*x;
        i=i+1;
    }
    printf total;
}
```

El fichero de salida es el siguiente:

TOK_PALABRA_RESERVADA	200	main	
TOK_SIMBOLO	300	{	
TOK_PALABRA_RESERVADA	201	int	
TOK_IDENTIFICADOR	500	x	
TOK_SIMBOLO	306	,	
TOK_IDENTIFICADOR	500	y	
TOK_SIMBOLO	302	;	
TOK_PALABRA_RESERVADA	201	int	
TOK_IDENTIFICADOR	500	i	
TOK_SIMBOLO	306	,	
TOK_IDENTIFICADOR	500	total	
TOK_SIMBOLO	302	;	
TOK_PALABRA_RESERVADA	213	scanf	
TOK_IDENTIFICADOR	500	x	
TOK_SIMBOLO	302	;	
TOK_PALABRA_RESERVADA	213	scanf	
TOK_IDENTIFICADOR	500	y	
TOK_SIMBOLO	302	;	
TOK_IDENTIFICADOR	500	i	
TOK_SIMBOLO	309	=	
TOK_NUMERO	501	1	
TOK_SIMBOLO	302	;	
TOK_IDENTIFICADOR	500	total	
TOK_SIMBOLO	309	=	
TOK_NUMERO	501	1	

```

TOK_SIMBOLO 302 ;
TOK_PALABRA_RESERVADA 211 while
TOK_SIMBOLO 307 (
TOK_IDENTIFICADOR 500 i
TOK_SIMBOLO 404 <=
TOK_IDENTIFICADOR 500 y
TOK_SIMBOLO 308 )
TOK_SIMBOLO 300 {
TOK_IDENTIFICADOR 500 total
TOK_SIMBOLO 309 =
TOK_IDENTIFICADOR 500 total
TOK_SIMBOLO 303 *
TOK_IDENTIFICADOR 500 x
TOK_SIMBOLO 302 ;
TOK_IDENTIFICADOR 500 i
TOK_SIMBOLO 309 =
TOK_IDENTIFICADOR 500 i
TOK_SIMBOLO 312 +
TOK_NUMERO 501 1
TOK_SIMBOLO 302 ;
TOK_SIMBOLO 301 }
TOK_PALABRA_RESERVADA 214 printf
TOK_IDENTIFICADOR 500 total
TOK_SIMBOLO 302 ;
TOK_SIMBOLO 301 }

```

Autocorrección de la práctica

En la página web del laboratorio de Procesadores de Lenguaje hay disponible un enlace para que el alumno se descargue el fichero comprimido **autocorreccion_lexico.zip**. Este fichero contiene 10 programas escritos en ALFA y sus correspondientes 10 ficheros de salida que debe generar el programa **prueba_lexico** desarrollado en esta primera práctica. El alumno debe comparar las salidas que obtiene de su programa **prueba_lexico** con las salidas proporcionadas en el fichero **autocorreccion_lexico.zip**. Para realizar la comparación en Linux se puede utilizar el comando **diff** que compara el contenido de dos ficheros.

Gramática del Lenguaje de Programación ALFA

```

1  <programa> ::= main { <declaraciones> <funciones> <sentencias> }
2  <declaraciones> ::= <declaracion>
3  | <declaracion> <declaraciones>
4  <declaracion> ::= <clase> <identificadores> ;
5  <clase> ::= <clase_escalar>
6  | <clase_puntero>
7  | <clase_vector>
8  | <clase_conjunto>
9  <clase_escalar> ::= <tipo>
10 <tipo> ::= int
11 | boolean
12 | float
13 <clase_puntero> ::= <tipo> *
14 | <clase_puntero> *
15 <clase_vector> ::= array <tipo> [ <constante_entera> ]
16 | array <tipo> [ <constante_entera> , <constante_entera> ]
17 <clase_conjunto> ::= set of <constante_entera>
18 <identificadores> ::= <identificador>
19 | <identificador> , <identificadores>
20 <funciones> ::= <funcion> <funciones>
21 |
22 <funcion> ::= function <tipo> <identificador> ( <parametros_funcion> ) {
  <declaraciones_funcion> <sentencias> }

```

23	<parametros_funcion>	::=	<parametro_funcion> <resto_parametros_funcion>
24			
25	<resto_parametros_funcion>	::=	; <parametro_funcion> <resto_parametros_funcion>
26			
27	<parametro_funcion>	::=	<tipo> <identificador>
28	<declaraciones_funcion>	::=	<declaraciones>
29			
30	<sentencias>	::=	<sentencia>
31			<sentencia> <sentencias>
32	<sentencia>	::=	<sentencia_simple> ;
33			<bloque>
34	<sentencia_simple>	::=	<asignacion>
35			<lectura>
36			<escritura>
37			<liberacion>
38			<retorno_funcion>
39			<operacion_conjunto>
40	<bloque>	::=	<condicional>
41			<bucle>
42			<seleccion>
43	<asignacion>	::=	<identificador> = <exp>
44			<elemento_vector> = <exp>
45			<acceso> = <exp>
46			<identificador> = malloc
47			<identificador> = & <identificador>
48	<elemento_vector>	::=	<identificador> [<exp>]
49			<identificador> [<exp> , <exp>]
50	<condicional>	::=	if (<exp>) { <sentencias> }
51			if (<exp>) { <sentencias> } else { <sentencias> }
52	<bucle>	::=	while (<exp>) { <sentencias> }
53			for (<identificador> = <exp> ; <exp>) { <sentencias> }
54	<lectura>	::=	scanf <identificador>
55			scanf <elemento_vector>
56	<escritura>	::=	printf <exp>
57			cprintf <identificador>
58	<liberacion>	::=	free <identificador>
59	<acceso>	::=	* <identificador>
60			* <acceso>
61	<retorno_funcion>	::=	return <exp>
62	<seleccion>	::=	switch (<exp>) { <casos_seleccion> }
63	<casos_seleccion>	::=	<casos_estandar> <caso_defecto>
64	<casos_estandar>	::=	<caso_estandar>
65			<casos_estandar> <caso_estandar>
66	<caso_estandar>	::=	case <constante_entera> : <sentencias>
67	<caso_defecto>	::=	default <sentencias>
68	<operacion_conjunto>	::=	union (<identificador> , <identificador> , <identificador>)
69			intersection (<identificador> , <identificador> ,
70			<identificador>)
71			add (<exp> , <identificador>)
72	<exp>	::=	clear (<identificador>)
73			<exp> + <exp>
74			<exp> - <exp>
75			<exp> / <exp>
76			<exp> * <exp>
77			- <exp>
78			<exp> && <exp>
79			<exp> <exp>
80			! <exp>
81			<identificador>
82			<constante>
			(<exp>)

```

83      | ( <comparacion> )
84      | <acceso>
85      | <elemento_vector>
86      | size ( <identificador> )
87      | contains ( <exp> , <identificador> )
88      | <identificador> ( <lista_expresiones> )
89 <lista_expresiones> ::= <exp> <resto_lista_expresiones>
90      |
91 <resto_lista_expresiones> ::= , <exp> <resto_lista_expresiones>
92      |
93 <comparacion> ::= <exp> == <exp>
94      | <exp> != <exp>
95      | <exp> <= <exp>
96      | <exp> >= <exp>
97      | <exp> < <exp>
98      | <exp> > <exp>
99 <constante> ::= <constante_logica>
100      | <constante_entera>
101      | <constante_real>
102 <constante_logica> ::= true
103      | false
104 <constante_entera> ::= <numero>
105 <numero> ::= <digito>
106      | <numero> <digito>
107 <constante_real> ::= <constante_entera> . <constante_entera>
108 <identificador> ::= <letra>
109      | <letra> <cola_identificador>
110 <cola_identificador> ::= <alfanumerico>
111      | <alfanumerico> <cola_identificador>
112 <alfanumerico> ::= <letra>
113      | <digito>
114 <letra> ::= a | b | ... | z | A | B | ... | Z
115 <digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

Consideraciones Adicionales

- El lenguaje permite incluir comentarios entre los caracteres // y el final de la línea (son comentarios de una sola línea).
- Los identificadores se limitan a una longitud de 50 caracteres.