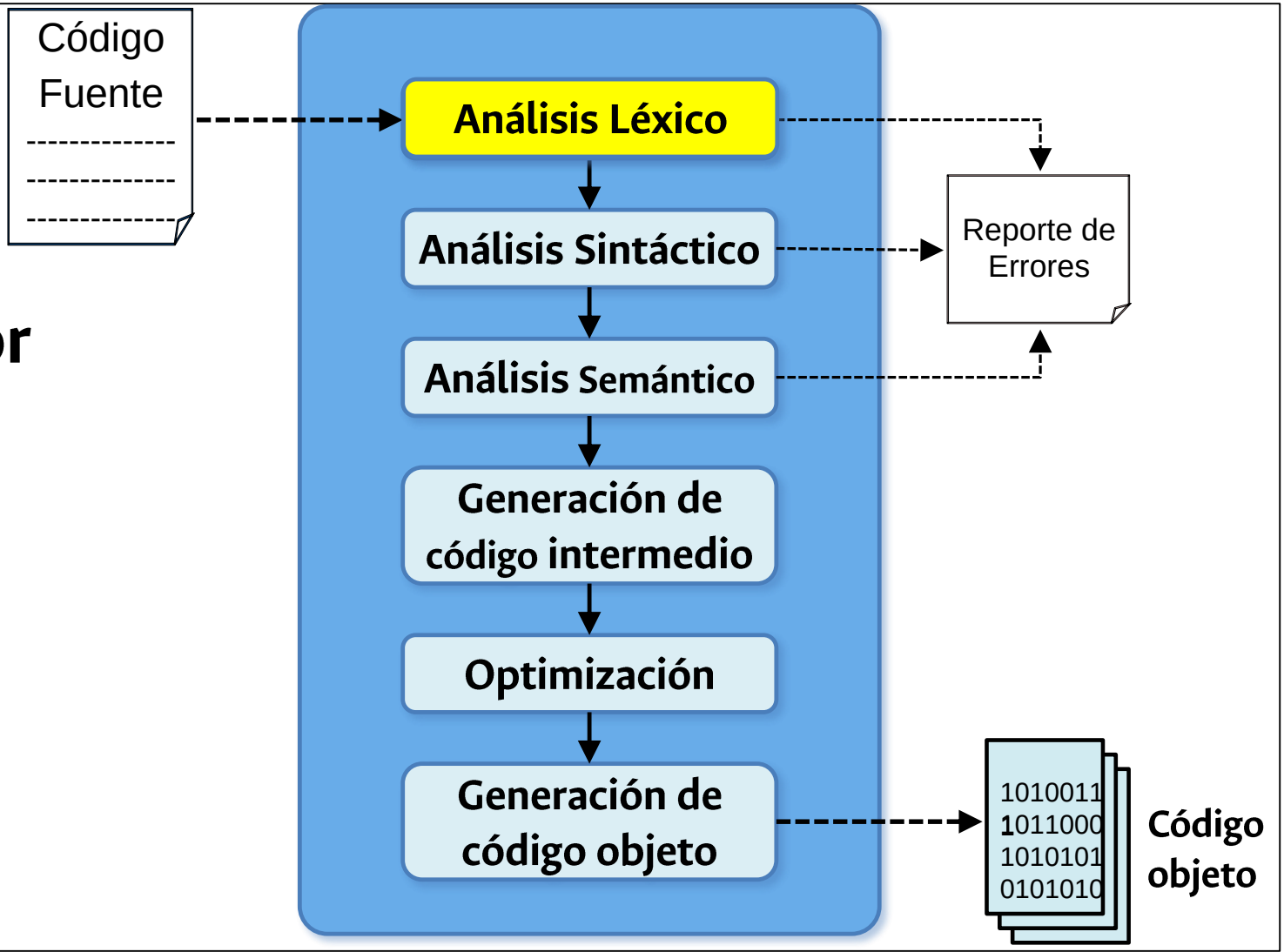
UNIVERSIDAD
NACIONAL
DE COLOMBIA

Análisis Léxico

Felipe Restrepo Calle
ferestrepoca@unal.edu.co

Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia
Sede Bogotá

Fases de un compilador



- Analizador léxico (**scanner/tokenizer/lexer**).
- Recibe un **archivo de entrada (código fuente)**
- **Agrupa** los caracteres del archivo en elementos léxicos: **tokens ("palabras")**
- Comúnmente se centra en elementos importantes y descarta otros elementos del lenguaje que no son importantes para el análisis: espacios en blanco, tabulaciones, caracteres de nueva línea, comentarios, ...

  Ejemplo: 1. "6-2*30/7"
  2. "6 - 2 * 30/ 7"

  Estructura equivalente pero usan distintos caracteres

Errores léxicos:

- Caracteres que no pertenecen al alfabeto del lenguaje (“**$**”, “**Ñ**”, “**°**”)
- Cadena que no coincide con ninguno de los patrones de los tokens posibles (p.e. en un lenguaje donde “**:=**“ es la asignación pero “**:**” no puede aparecer solo)
- Caracteres no permitidos en un contexto determinado (“**.23**”)

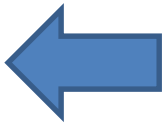
En este punto el compilador tiene una visión muy local. Por ejemplo, si ve la cadena **“wihle”** creerá que es un identificador, cuando posiblemente se trata de un **“while”** mal escrito. Se informará del error en una etapa posterior y no en el análisis léxico.

- La cadena de caracteres concreta que representa un **token** se denomina **lexema.**
- El lexema no juega un papel desde el punto de vista estructural, pero sí desde el semántico.
- A la información auxiliar que acompaña a un token se le llama **atributos del token** (lexema, fila, columna,. . .).

Análisis Léxico

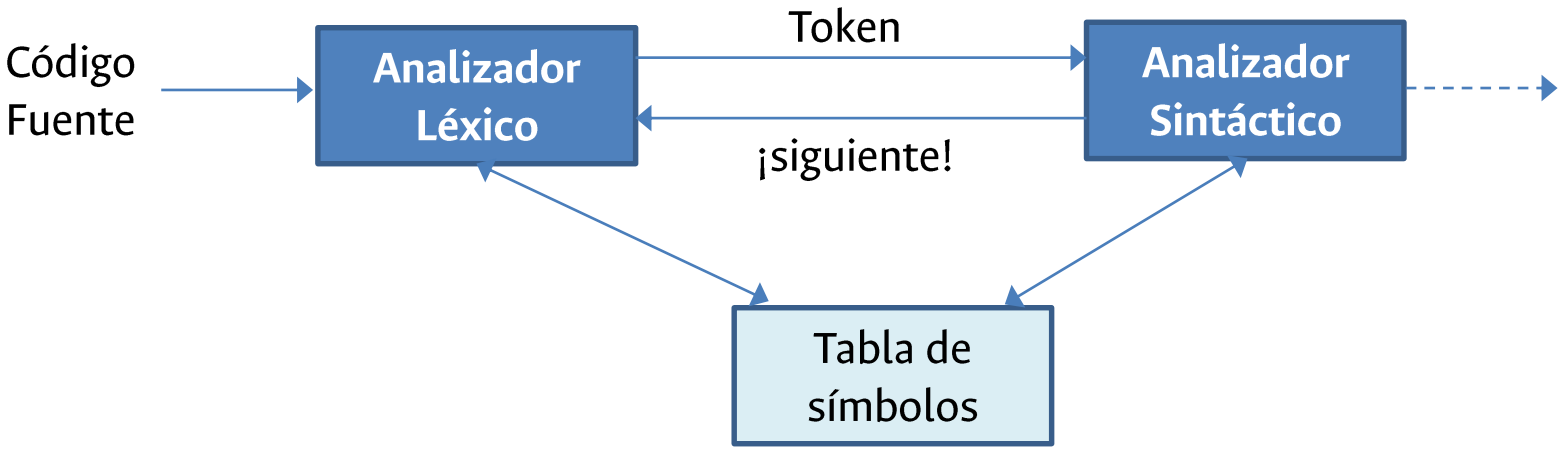
Ejemplo: "6-2.5*30/divisor"

Entero, 6
Resta, -
Real, 2.5
Multiplicación, *
Entero, 30
División, /
Identificador, divisor



Cada elemento léxico:
Token, Lexema

Operaciones durante el análisis léxico:



- Procesado léxico del programa fuente (tokens, lexemas, interacción con A. Sintáctico)
- Manejo del archivo de código fuente (abrir, leer caracteres, cerrar, gestión de errores de lectura de archivo)
- Ignorar comentarios y , en lenguajes de formato libre, ignorar separadores (espacios en blanco, tabuladores, retornos de carro, etc.)
- Localización de errores (número de línea, posición).
- Preproceso de macros, definiciones, constantes, inclusión de otros archivos, ...

- **Clases de tokens**:
  - palabras reservadas (if, then,. . .)
  - símbolos especiales (operadores aritméticos, lógicos,. . .)
  - cadenas no específicas (identificador, número real,. . .)
  - **EOF** (fin de archivo)

- Se especifican mediante **expresiones regulares**

Expresión Regular	Token
[a-z][a-z0-9]*	id
if	if
[0-9]+	entero
>	mayor
>=	mayor_igual
/	div

Consideraciones:

- Política de elección de palabra reservada sobre identificador
- Principio de la subcadena más larga
- Con algunos tokens nos tenemos que “pasar” leyendo la entrada (es necesario leer uno o varios caracteres más para determinar de qué token se trata)

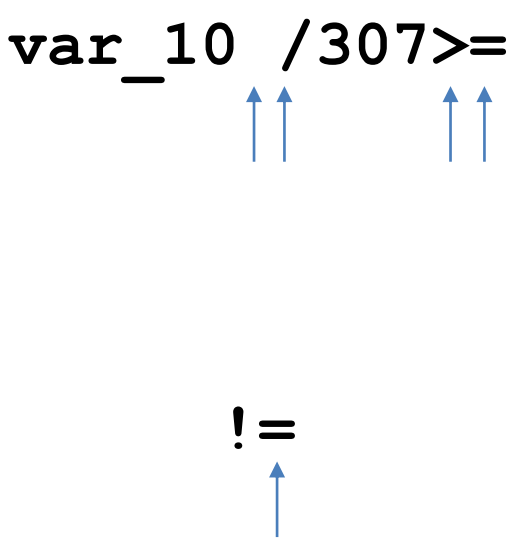
Identificación de palabras reservadas

Palabra reservada vs. identificador

- **Resolución implícita:** tras reconocer un identificador en el DT, se consulta en una lista de palabras reservadas (o en la tabla de símbolos) para decidir el token a devolver

- **Resolución explícita:** se indican todas las expresiones regulares de todas las palabras reservadas y se integran en los DT (es necesario si se usa un generador automático de analizadores léxicos: Flex, ANTLR, ...)

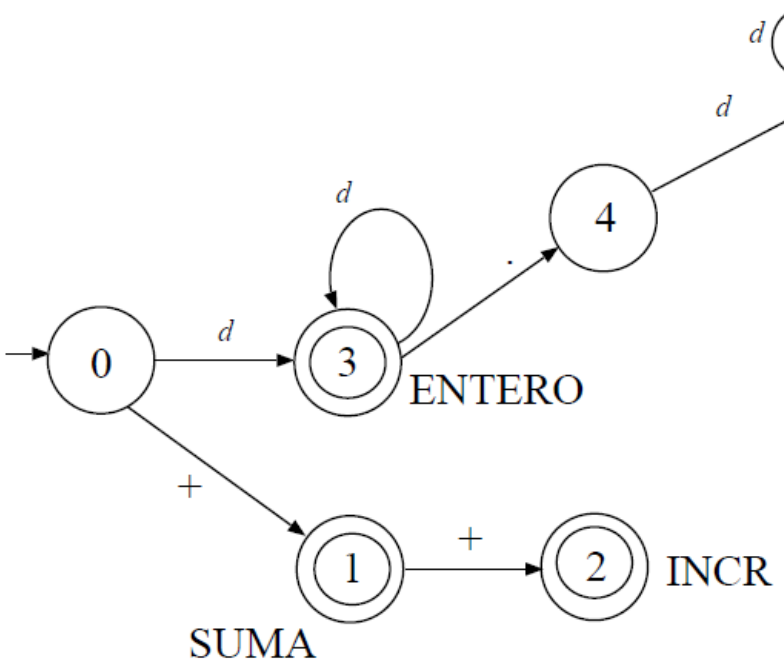
Ejemplo de procesamiento:



Análisis Léxico

¿Cómo se especifica? → Autómatas Finitos Deterministas (simplificados)

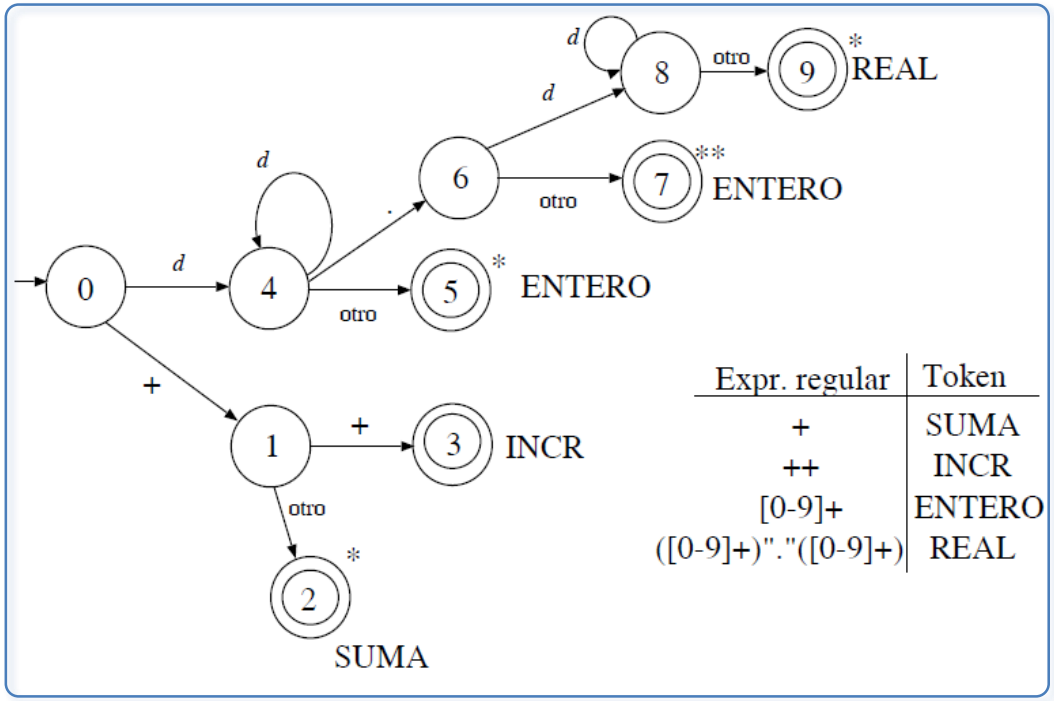
Ejemplo



Expr. regular	Token
+	SUMA
++	INCR
[0-9]+	ENTERO
([0-9]+)"."([0-9]+)	REAL

Análisis Léxico

¿Cómo se especifica? → Diagrama de transiciones



Expr. regular	Token
+	SUMA
++	INCR
[0-9]+	ENTERO
([0-9]+)"."([0-9]+)	REAL

- En el DT se especifica lo que debe hacer el analizador léxico con cada carácter de la entrada
- Algunos estados finales están marcados con uno o más asteriscos; esto indica que se debe devolver al buffer de entrada tantos caracteres como asteriscos tenga el estado

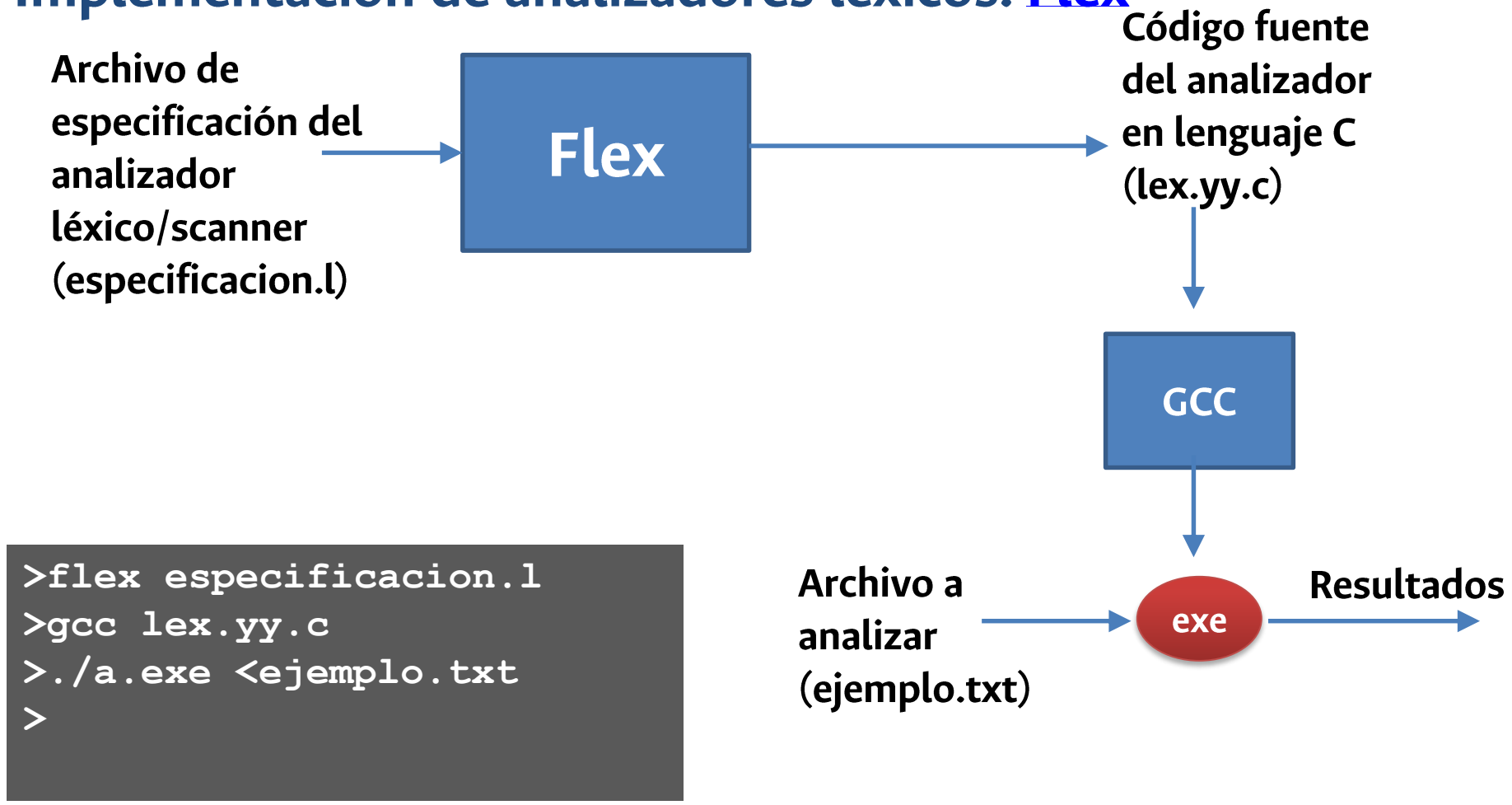
- De los estados finales no sale ninguna transición
- En algunos casos (números, id) es necesario leer un carácter más para estar seguros de haber reconocido el token completo

Implementación de analizadores léxicos

- **Usando un generador automático:** Flex, ANTLR, ...

- **"A mano":**
  - ✓ implementando el diagrama de transiciones
  - ✓ implementando directamente el analizador usando estructuras de selección múltiple (switch)

Implementación de analizadores léxicos: Flex



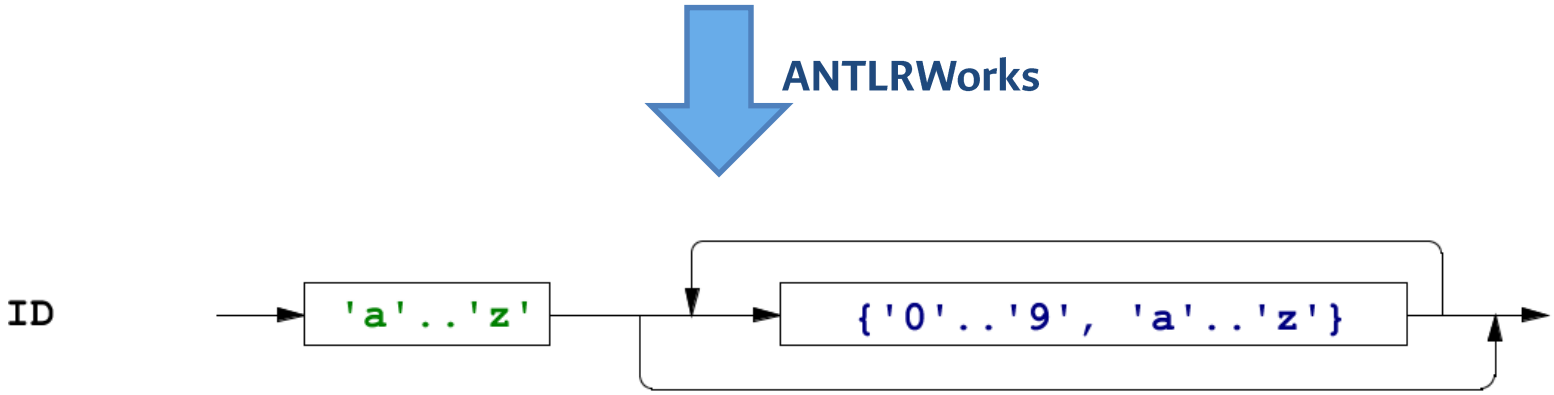

```
>flex especificacion.l
>gcc lex.yy.c
>./a.exe <ejemplo.txt
>
```

- **FLEX:**

```
[a-z][a-z0-9]* { return ID;}
```

- **ANTLR:**

```
ID : ('a'..'z')('a'..'z'|'0'..'9')*;
```



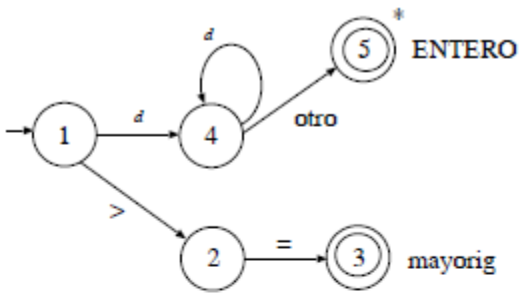
Implementación de analizadores léxicos:

Representación de los tokens

```
public class Token {

    public int fila;
    public int columna;

    public String lexema;

    public int tipo;    // tipo es: ID, ENTERO, REAL ...

    public static final int
        ID      = 1,
        ENTERO  = 2,
        REAL    = 3,
        ...
...
}
```

Implementación de analizadores léxicos:

implementando DT




```
int delta (int estado, int c) {
  switch (estado) {
    case 1: if (c==’>’) return 2;
            else if (c>=’0’ && c<=’9’) return 4;
            else return -1;
            break;
    case 2: if (c==’=’) return 3;
            else return -1;
            break;
    case 3: return -1;
            break;
    case 4: if (c>=’0’ && c<=’9’) return 4;
            else return 5;
            break;
    case 5: return -1;
            break;
    default: /* error fatal */
  }
}
```

Implementación de analizadores léxicos:

Diferenciación manual de tokens en un switch

```
1   int analex()
2   {
3       c = obtenercaracter();
4       switch (c)
5       {
6           case ' ':
7           case '\t':
8           case '\n':  /* Y para los demás separadores, no hacer nada */
9               break;
10
11          case '+':
12          case '-': return(ADDOP);
13
14          case '*':
15          case '/': return(MULOP);
16
17          /* Y el resto de operadores y elementos de puntuación */
```

Implementación de analizadores léxicos:

Diferenciación manual de tokens en un switch

```
17          /* Y el resto de operadores y elementos de puntuación */
18
19          default:
20              if (ESNUMERO(c))
21              {   /* leer caracteres mientras sean números */
22                  /* devolver a la entrada el último caracter leído +/
23                  /* almacenar el lexema leído */
24                  return (NUNINT);
25              }
26              else if (ESLETRA(c))
27              {   /* leer caracteres mientras sean letras o números */
28                  /* devolver al buffer de entrada el último caracter leído +/
29                  /* comprobar si es una palabra reservada */
30                  /* almacenar el lexema leído */
31                  return (token);      // que será una palabra reservada o un identificador
32              }
33      }   // del switch
34  } // de analex
35
```