

---

# Análisis Léxico

---

Diseño de Lenguajes de Programación

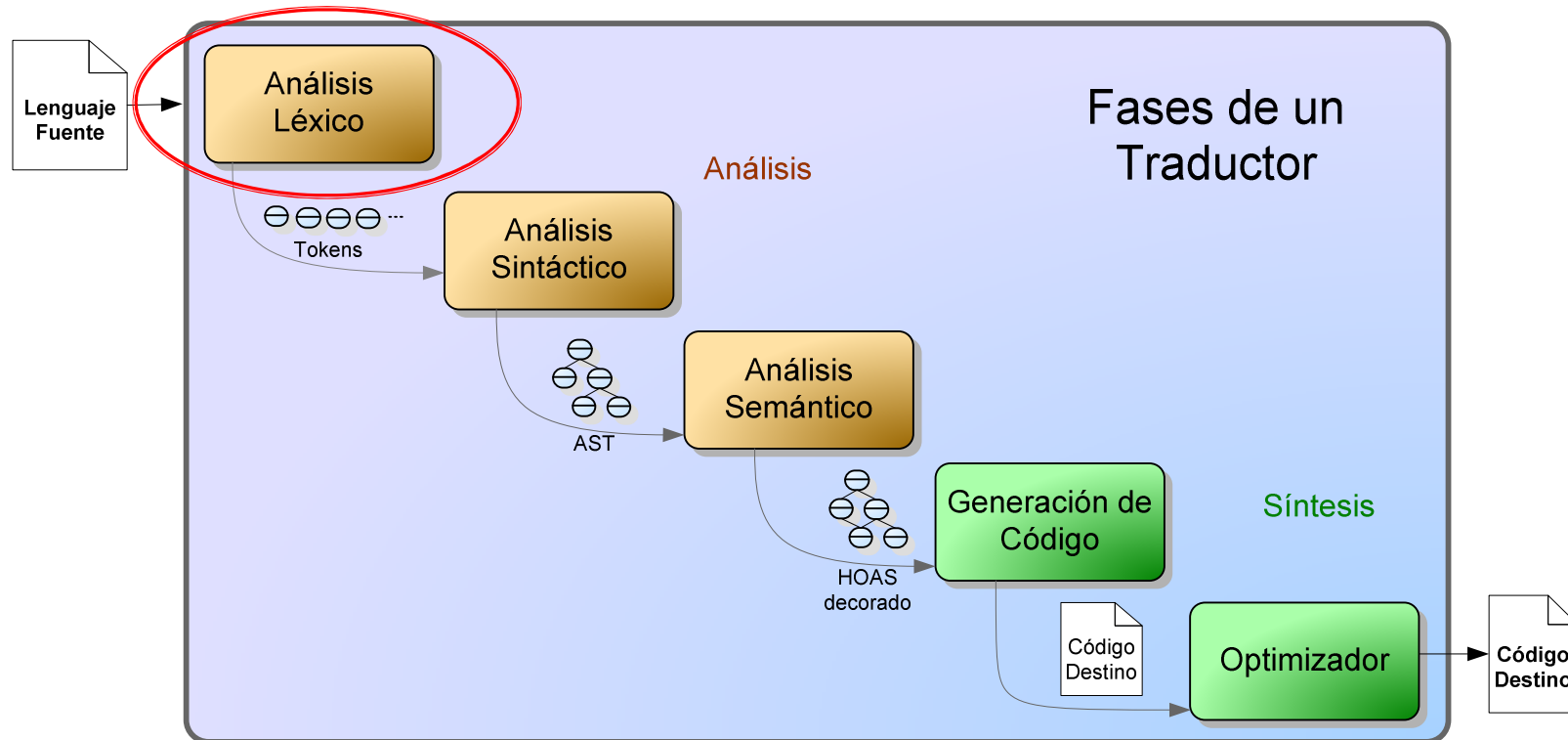
Ingeniería Informática

Universidad de Oviedo

(v1.13)

Raúl Izquierdo Castanedo

# Fases de un Traductor



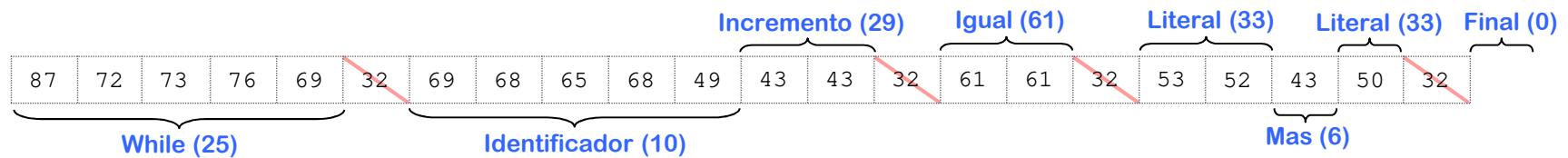
# Funciones

## Funciones

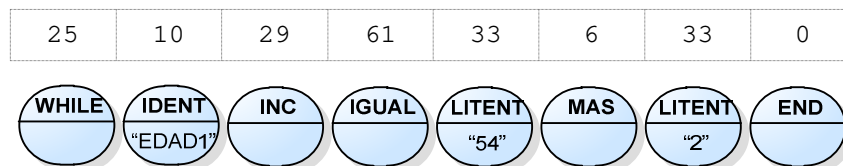
- Ignorar caracteres no relevantes
- Detectar cadenas que no pertenezcan al lenguaje
- Separar lexemas y clasificarlos en Tokens (categorías)

## Ejemplo

- Entrada: 22 números (caracteres)



- Salida: 8 números (tokens)



# Lexema, Token y Patrón

## Definiciones

- Lexema

- Agrupación de caracteres que constituyen los símbolos con los que se forman las sentencias del lenguaje

12            386            contador            getEdad

- Token

- Conjunto de lexemas que puede ser tratado como una unidad sintáctica

LITERAL    LITERAL    IDENTIFICADOR            IDENTIFICADOR

- Criterio

- ¿Delimitadores? [ ] { }
- ¿Operadores? + - \* /

- Patrón

- Regla que permite determinar qué lexemas pertenecen a un token

Token: IDENTIFICADOR                            lexema: a32

- ¿En qué lenguaje se expresan los patrones?

---

# Metalinguajes

# Metalinguajes

## Definición de Patrones

- En lenguaje natural
  - Las constantes reales incluyen un punto (no exponente)
- Problemas del lenguaje natural
  - O poco detallado

`a = .5;`                      ¿Válido?

- O demasiado extenso

“An identifier is a sequence of letters and digits; the first character must be a letter. The underscore `_` counts as a letter. Upper- and lowercase letters are different. If the input stream has been parsed into tokens up to a given character, the next token is taken to include the longest string of characters that could possibly constitute a token. Blanks, tabs, newlines, and comments are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.”

- Metalinguajes
  - Descripción formal
    - Precisión
    - Concisión
  - Ventajas
    - Facilitan Comprensión
    - Facilitan Implementación

---

# Metalinguajes

## Por fase en la que se utilizan

- Léxicos
  - Autómatas Finitos
  - Expresiones Regulares
- Sintácticos
  - Diagramas Sintácticos
  - BNF
  - EBNF
- Semánticos
  - Gramáticas Atribuidas
- Generación del Código
  - Code Functions

---

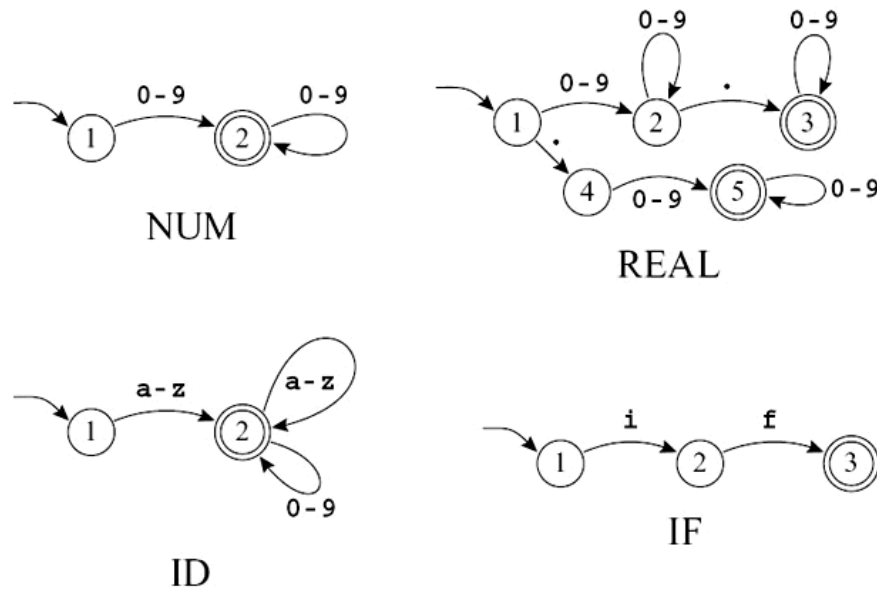
# Metalinguajes Léxicos



# Metalinguajes Léxicos

## Autómatas Finitos

- Paso 1. Definir cada token con un autómata



- Paso 2. Unir, hacer determinista y simplificar

.5      ¿Válido?

# Metalinguajes Léxicos

## Expresiones Regulares

### ■ Operadores

- ❑ Concatenación       $pa$
- ❑ Cierre: '\*'       $pa^*$
- ❑ Cierre Positivo: '+'       $pa^+$
- ❑ Alternativa: '|'       $p|a$
- ❑ Agrupación: '(' ')'       $(pa)^+$

### ■ Ejemplos

$(0|1|2|3|4|5|6|7|8|9)^+$   
 $(0|1|2|3|4|5|6|7|8|9)^* \cdot (0|1|2|3|4|5|6|7|8|9)^+$   
while

.5      ¿Válido?

# Interfaz del Léxico

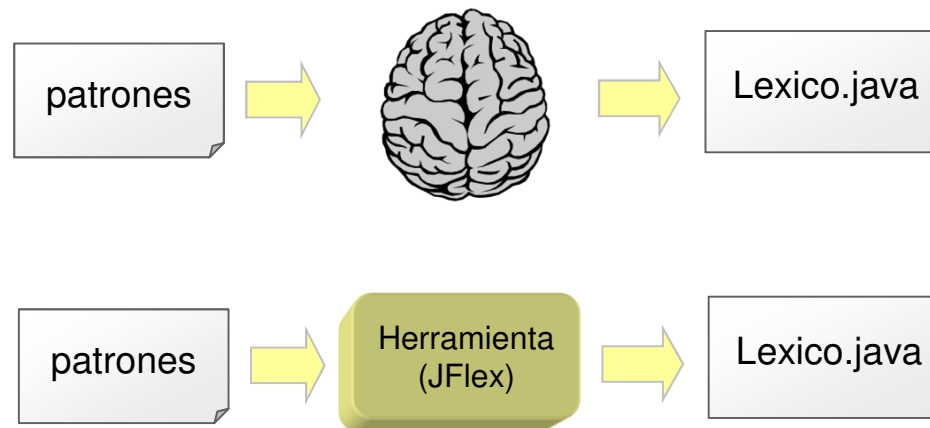
---

Procesadores de Lenguaje

# Construcción Analizador Léxico

## Pasos para obtener Analizador Léxico

- Determinar los tokens
- Definir un patrón para cada uno
- Implementar los patrones
  - Manual
  - Herramientas



---

# Funcionalidad Requerida

## **Independientemente de cómo esté implementado**

- Eliminar espacios y comentarios
- Control de número de línea
- Mismo interfaz de Entrada y de Salida

# Interfaz de Entrada

## ¿De dónde recibe el léxico los caracteres?

```
public class Léxico {  
    private Reader input;  
  
    public Léxico(Reader reader) {  
        input = reader;  
    }  
    ...  
}
```

### ■ Posibilidades

```
Léxico lex = new Léxico(new InputStreamReader(System.in));  
Léxico lex = new Léxico(new StringReader("cont = 12+x;"));  
Léxico lex = new Léxico(new FileReader("programa.txt"));  
Léxico lex = new Léxico(new FileReader(argv[0]));
```

---

# Interfaz de Salida

## **No se procesa todo el fichero de una vez**

- yylex

## **Valores a devolver**

- ¿Qué valores hay que devolver?
  - Generalmente son cuatro
- ¿De qué tipo es cada uno?
- ¿Cómo los devuelve el léxico?
  - Protocolo de Acceso

# Interfaz de Salida

## Representación de Tokens

- Se acuerda representación mediante constantes

```
public interface Tokens {  
    static final int IDENT = 1;  
    static final int IF = 2;  
    static final int LITENT = 3;  
    static final int LITREAL = 4;  
    static final int IGUAL = 5;  
    static final int PTOYCOMA = 6;  
}
```

- El 0 se reserva



# Interfaz de Salida

## Representación de Tokens

- Los tokens de un solo carácter utilizan su código ASCII

```
public interface Tokens {  
    static final int IDENT = 257;  
    static final int IF = 258;  
    static final int LIDENT = 259;  
    static final int LITREAL = 260;  
    static final int IGUAL = '=';           // Redundante  
    static final int PTOYCOMA = ';';       // Redundante  
}
```

- Facilita la lectura del sintáctico

```
asigna: IDENT IGUAL PARABRIR expr PARCERRAR PTOYCOMA  
asigna: IDENT '=' '(' expr ')' ';' ;
```

- Numeración comienza en 256
- No se definen como Tokens

# Interfaz de Salida

## Protocolo de Acceso

- Un método por valor

```
class Léxico {  
  
    int yylex() { ... }  
  
    String lexeme()  
    { ... }  
  
    int line() { ... }  
    ...  
}
```

- Un método único

```
class Léxico {  
    Token yylex() { ... }  
    ...  
}  
  
class Token {  
    int getToken() { ... }  
    String lexeme() { ... }  
    int line() { ... }  
}
```

En la práctica usaremos un sistema mixto:

- JFlex usa un método por valor...
- ... pero los usaremos para crear un objeto Token

# Interfaz. Resumen

## Interfaces de Entrada y de Salida

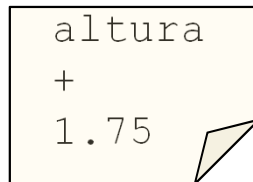
```
Léxico lex = new Léxico(new FileReader("prog.c"));
```

```
int token;
```

```
while ((token = lex.yylex()) != 0)
```

```
    Sop(lex.line()+":" + token + ":" + lex.lexeme());
```

### Entrada



altura  
+  
1.75

```
interface Tokens {  
    static final int IDENT = 257;  
    static final int IF = 258;  
    static final int LITENT = 259;  
    static final int LITREAL = 260;  
}
```

### Salida

	1	2	3	4
yylex				
lexeme				
line				

# Implementación Manual

---

Análisis Léxico

# Implementación Manual

## Pasos para obtener Analizador Léxico

- Determinar los tokens
- Definir un patrón para cada token
- Implementar los patrones



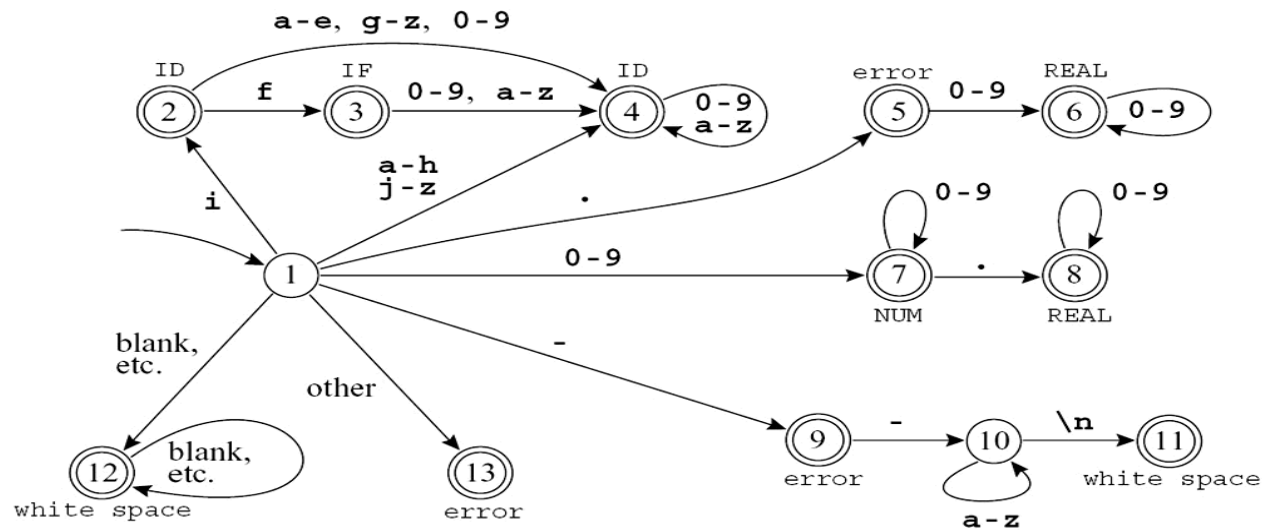
## Alternativas más comunes

- Tabla de estados
- Implementación Estructurada

# Tabla de Estados

## Proceso

- Se unen autómatas, se hace determinista y se simplifica



```
int autómatas[][]={ /* ... 0 1 2...+...e f g h i ... */
/* state 0 */      {0,0, ... 0,0,0...0...0,0,0,0,0 ...},
/* state 1 */      {0,0, ... 7,7,7...9...4,4,4,4,2 ...},
/* state 2 */      {0,0, ... 4,4,4...0...4,3,4,4,4 ...},
/* state 3 */      {0,0, ... 4,4,4...0...4,4,4,4,4 ...},
/* state 4 */      {0,0, ... 4,4,4...0...4,4,4,4,4 ...},
/* state 5 */      {0,0, ... 6,6,6...0...0,0,0,0,0 ...},
/* state 6 */      {0,0, ... 6,6,6...0...0,0,0,0,0 ...},
...
}
```

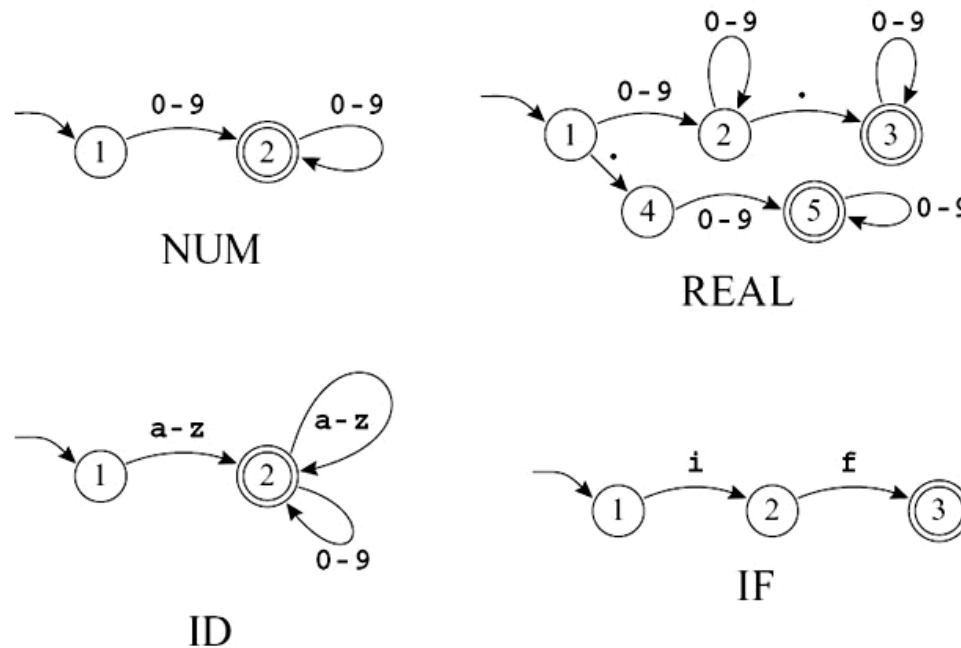
# Implementación Estructurada

## Cuando el autómata se asemeja a un ordinograma

- Estructuras secuencial, alternativa e iterativa

## La implementación es la traducción a código del diagrama

- Es la implementación más eficiente

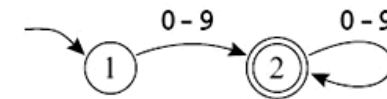


# Implementación Estructurada

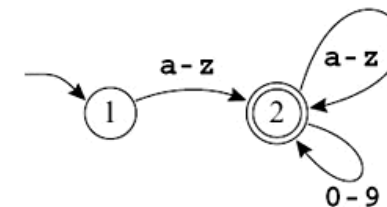
```
public int yylex() throws IOException {
    while (true) {
        while (Character.isWhitespace(getChar()))
            readNext();

        if (noMoreChars()) return 0;
        if (getChar() == ';') { readNext(); return ';'; }
        if (getChar() == '=') { readNext(); return '='; }

        if (Character.isDigit(getChar())) {
            StringBuffer buffer = new StringBuffer();
            buffer.append(getChar());
            while (Character.isDigit(readNext()))
                buffer.append(getChar());
            lexeme = buffer.toString();
            return Tokens.LITENT;
        }
        if (Character.isLetter(getChar())) {
            StringBuffer buffer = new StringBuffer();
            buffer.append(getChar());
            while (Character.isLetterOrDigit(readNext()))
                buffer.append(getChar());
            lexeme = buffer.toString();
            return Tokens.IDENT;
        }
        System.out.println("Error léxico: " + getChar());
        readNext();
    }
}
```



NUM



ID



# Generación Automática

---

Análisis Léxico

# Generación Automática

## Pasos para obtener un Analizador Léxico con una herramienta

- Determinar los tokens
- Definir un patrón para cada uno
- Escribir los patrones en el formato de la herramienta



`java -jar jflex.jar lexico.l`

# Avance del uso de la herramienta

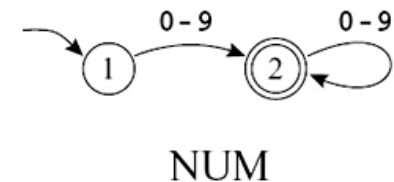
## Comparativa: reconocer un literal entero de ambas formas

### ■ Implementación manual

```
if (Character.isDigit(getChar())) {  
    StringBuffer buffer = new StringBuffer();  
    buffer.append(getChar());  
    while (Character.isDigit(readNext()))  
        buffer.append(getChar());  
    lexeme = buffer.toString();  
    return Tokens.LITENT;  
}
```

### ■ Implementación con herramienta

`[0-9]+`                      { *return Tokens.LITENT;* }



# Resumen [muy breve] del Manual de JFlex

---

---

# Formato de Entrada

## Fichero de entrada a JFlex

<Código de Usuario>

%%

<Opciones y Declaraciones>

%%

<Sección de Reglas>

// Comentarios al estilo Java en todas las secciones

# Sección de Código de Usuario

```
import z.x;
```

```
%%
```

```
<Opciones y Declaraciones>
```

```
%%
```

```
<Sección de Reglas>
```

```
import z.x;
```

```
class Yylex {
```

```
    int yylex() {
```

```
    }
```

```
}
```

# Sección de Opciones y Declaraciones (I)

## Código de Clase

<Código de Usuario>

%%

%{

int n;

void f(void) {

...

}

%}

%%

<Sección de Reglas>

class **Yylex** {

int n;

void f(void) {

...

}

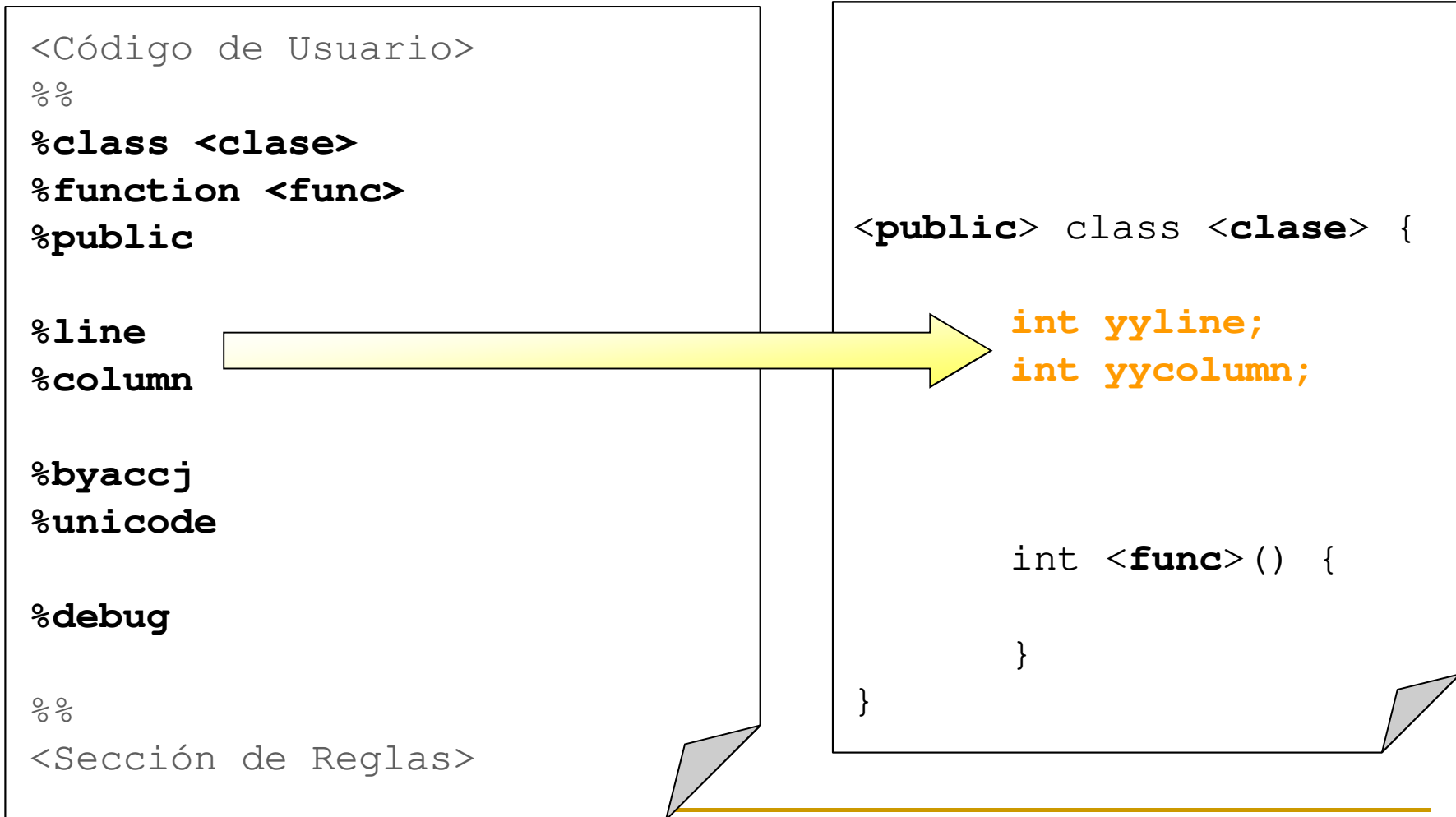
int **yylex**() {

}

}

## Sección de Opciones y Declaraciones (II)

### Directivas para controlar el código generado





# Sección de Reglas

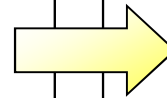
<Código de Usuario>

%%

<Opciones y Declaraciones>

%%

[patrón 1]	[acción 1]
[patrón 2]	[acción 2]
[0-9]+	{ return ENTERO; }



class **Yylex** {

int **yylex**() {  
...  
[acción1]  
...  
[acción2]  
...  
return ENTERO;  
...  
}

Tabla de estados

*!Error si la entrada no cumple ningún patrón!*

# Expresiones Regulares

Operador	Ejemplo	Sentencias
a   b		
a b		
a*		
a+		
(a)		
a?	pa?	p, pa
~a	p~a	pa, pba, pca, pbcdfsgea

# Expresiones Regulares

Operando	Ejemplo	Sentencias
<b>Sec. Escape</b>	\r \n \t a\+	a+
<b>Cadenas</b>	"a+" "a?b+" "a\n"	a+
<b>Conjuntos</b>	[abcde] [a-e] [a-zA-Z_] [+~*/]	a, b, c, d, e → (a b c d e) a, b, c, d, e letra o subrayado ¿?
<b>Conjunto Negado</b>	[^abc]	d, e, f, g, ...
<b>Punto</b>	p.a .+ (. \n)+	paa, pba, p4a, p\$a, ...

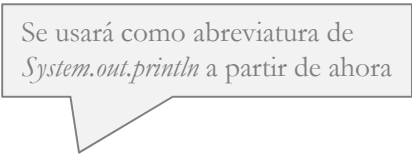
# Acciones

## Miembros Generados en la clase Yylex

```
Class Yylex {  
  
    int yylex() { ... }  
    String yytext() { ... }  
  
    private int yyline;           // Si %line  
    private int yycolumn;        // Si %column  
    ...  
}
```

## Ejemplo

```
%%  
%byaccj  
%unicode  
%line  
%%  
[a-zA-Z]+      { sop("Ident: " + yytext() + " en " + yyline); }  
.|\\n          { }
```



Se usará como abreviatura de *System.out.println* a partir de ahora

# Finalización de yylex

- ▣ **yylex finaliza al procesar todo el fichero**
  - Continúa después de ejecutar una *acción*

```
[a-z]+ { sop("1"); }  
[0-9]+ { sop("2"); }  
.|\\n { }
```

---

```
void main(String[] args) throws IOException  
{  
    Yylex lex;  
    lex = new Yylex(new FileReader("f.txt"));  
  
    lex.yylex() // Una llamada  
    // Ha ejecutado todas las acciones cuyo  
    // patrón se haya cumplido  
}
```

- ▣ **Para indicar que se desea salir de yylex al ejecutar una acción...**

```
[a-z]+ { sop("1"); return Tokens.IDENT; }  
[0-9]+ { sop("1"); return Tokens.LITENT; }  
.|\\n { } // En esta no finaliza
```

---

```
void main(String[] args) throws IOException  
{  
    Yylex lex;  
    lex = new Yylex(new FileReader("f.txt"));  
  
    int token;  
    while ((token = lex.yylex()) != 0)  
        sop(token);  
}
```

# Resolución de Conflictos

## Ejemplo

<code>[0-8]+</code>	<code>{ sop("regla 1"); }</code>
<code>[0-9]+</code>	<code>{ sop("regla 2"); }</code>

## Entrada

<b>32954</b>	→	regla 1
<b>32954</b>	→	regla 2

## Entrada

<b>32A</b>	→	regla 1
<b>32A</b>	→	regla 2

# Ejemplos

## Ejemplo

- Convertir un texto en mayúsculas y eliminar espacios repetidos

```
%%  
%byaccj  
%unicode  
%%  
[a-z]      { Sop(yytext().toUpperCase()); }  
" "+      { Sop(" "); }  
.|\\n      { Sop(yytext()); }  

```

## Ejemplo

- Contar caracteres, palabras y líneas

```
%%  
%byaccj  
%unicode  
%{  
    public int caracteres, palabras, lineas;  
}%  
%%  
[a-zA-Z][a-zA-Z0-9_]*      { palabras++; caracteres+= yytext().length(); }  
\\n                        { lineas++; caracteres++; }  
.  
    { caracteres++; }
```

---

# Ejercicio E1

## **Léxico del siguiente lenguaje**

```
edad = 65;  { Comentario de varias líneas }
```



---

# Tarea

## Mejoras sobre el léxico anterior

- Añadir más operadores
  - +, - \* / % { } ) [ ] “.” “,”
- Añadir palabras reservadas
  - return, if, while, else, return.
- Contar número de líneas

# Ejemplo

## Analizar línea de comando

```
c:\>prog -v -verbose -f <file> -file <file> -? -h -help
```

## Solución

```
%%
%byaccj
%unicode
%{
    public int verbose = false;
    public String file;
}%
%%
-h | "-?" | -help          { sop("Ayuda: -v -f <file>");}

-v | -verbose              { verbose = true; }

-f" "[a-zA-Z0-9_\.]+       { file = yytext().substring(3);}
-file" "[a-zA-Z0-9_\.]+    { file = yytext().substring(6);}

" " { }

. { sop("Opción no permitida"); }
```

# Solución E1

```
edad = 65; { Comentario de varias líneas }
```

```
interface Tokens {  
    static final int IDENT = 257;  
    static final int LITENT = 258;  
}
```

```
%%  
%byaccj  
%unicode  
%line  
%column  
  
%{  
    public string lexeme() { return yytext(); }  
    public int line() { return yyline+1; }  
    public int column() { return yycolumn+1; }  
%}  
  
%%  
[=;]      { return yytext().charAt(0); }      // yycharat(0)  
  
[a-zA-Z][a-zA-Z0-9_]* { return Tokens.IDENT; }  
[0-9]+     { return Tokens.LITENT; }  
  
[ \n\r\t]      { }  
"{ " ~"}"      { }      // Normalmente patrones simples  
  
.      { Sop("Error " + line() + ":" + column() + " Lexema = " + yytext());}
```