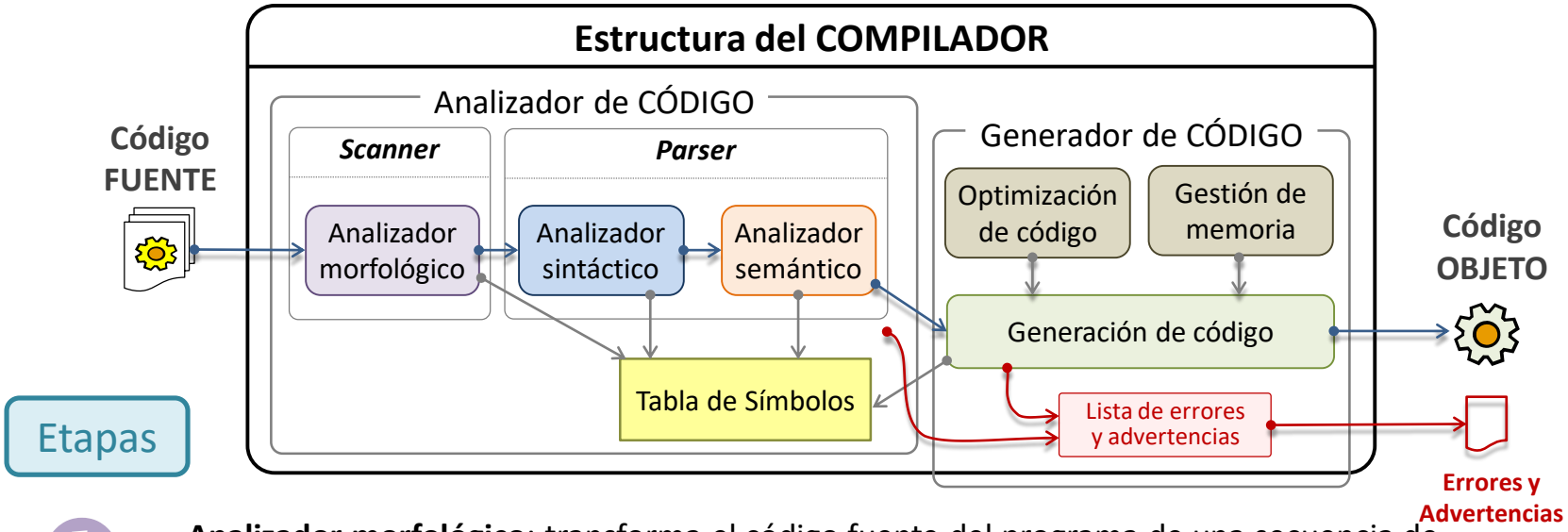
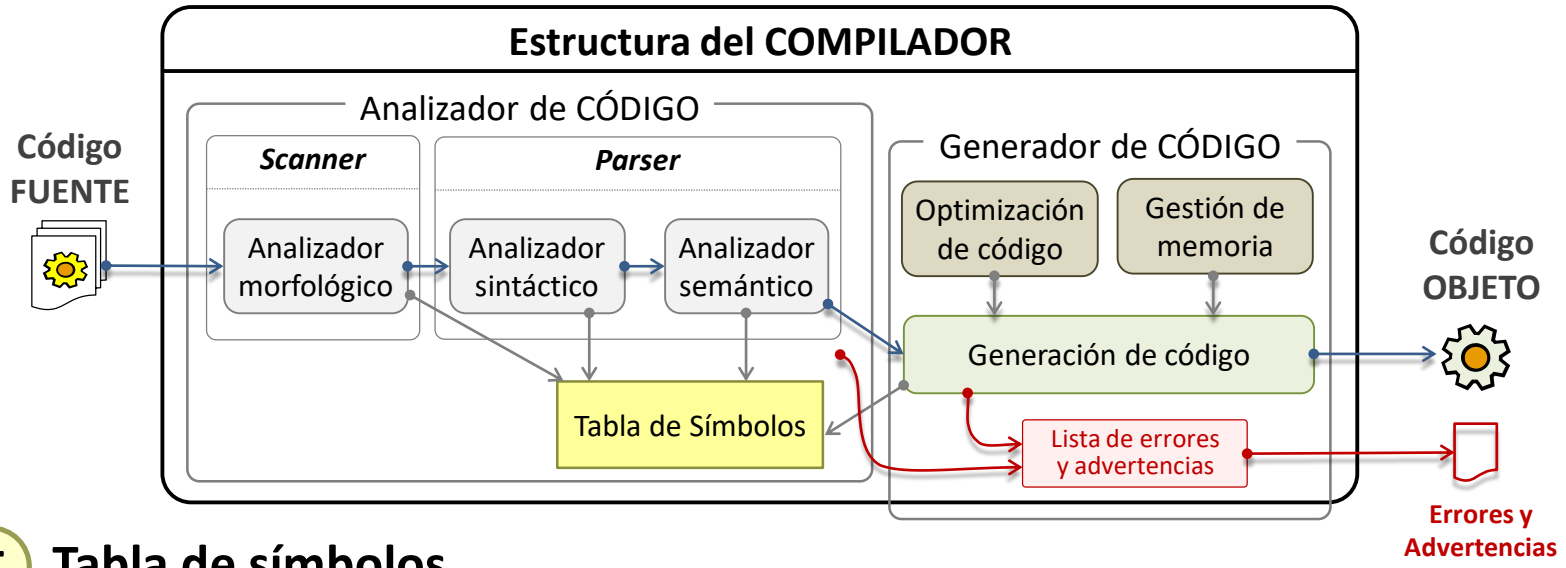
Índice

- **Tabla de Símbolos**
  - Tabla de Símbolos para el lenguaje ALFA
  - Diseño de información de la Tabla de Símbolos para el lenguaje ALFA
  - Programa de prueba para la TABLA DE SÍMBOLOS para el lenguaje ALFA
- **Analizador Morfológico**
  - Flex: herramienta para la creación de Análisis Morfológico
  - Instalación, compilación y generación de Flex



1. **Analizador morfológico**: transforma el código fuente del programa de una secuencia de caracteres, a una secuencia de unidades sintácticas (tokens).
2. **Analizador sintáctico**: interpreta las unidades sintácticas identificadas (tokens) por el analizador morfológico como un programa estructurado según una gramática (ya se verá en Autómatas y Lenguajes).
3. **Analizador semántico**: realiza comprobaciones semánticas como por ejemplo que las variables están declaradas antes de su uso, que los tipos de las expresiones son correctos, etc.

T **Tabla de símbolos o identificadores**. Se encarga de todos los aspectos dependientes del contexto relacionados con los *nombres* (variables, constantes, funciones, palabras reservadas, etc.) que puedan aparecer en los programas.



T Tabla de símbolos

- La tabla de símbolos, es la parte en la que se guarda la información necesaria para poder determinar si el programa es correcto y (si lo es) generar el código.
- Esta información se centra en los **identificadores** que aparecen en el programa fuente (variables, funciones, procedimientos, etiquetas, etc.). Ejemplos delo qu3e se suela guardar de cada uno de ellos:
  - Tipo del identificador
  - Ámbito de validez del identificador
  - Si el identificador corresponde a una función, el número y tipo de sus argumentos.
  - Si el identificador corresponde a un vector, el tamaño del mismo, etc.

TABLA DE SÍMBOLOS para el lenguaje ALFA

- Se usará **estructura de datos** basada en tablas hash para implementar la tabla de símbolos del lenguaje ALFA
- La **estructura de datos** debe almacenar lo siguiente:
  - Las **variables** (globales del programa principal y locales de las funciones)
  - Los **parámetros** de las funciones
  - Las **funciones**
- Algunas características de diseño de tabla de símbolos del lenguaje ALFA a debe tener en cuenta:
  - Define vectores. Es necesario guardar el tamaño del vector
  - Variable inicializada. Guardar el valor de la variable para saber que ha sido inicializada
  - Parámetros de funciones. Guardar el número de parámetros para validar la llamadas
  - Tipo de variables. Comprobar la asignación de variables que sean del mismo tipo

Información de la TABLA DE SÍMBOLOS para ALFA

- **Clave de acceso**: Identificador (lexema) de la variable, parámetro o función.
- **Categoría del elemento** (variable, parámetro de función y función): define la categoría del identificador que está almacenado en la tabla de símbolos. A tener en cuenta:

```
#define VARIABLE 1
#define PARAMETRO 2
#define FUNCION 3
```

- **Tipo básico de dato** (boolean, int): para variables, parámetros de funciones, vectores y tipo de datos de retorno de función. A tener en cuenta:

```
#define BOOLEAN 1
#define INT 2
```

- **Clase** (escalar, vector): identifica la estructura de la información asociada al identificador de una variable o un parámetro. A tener en cuenta:

```
#define ESCALAR 1
#define VECTOR 2
```

- **Tamaño** (número de filas, valores 1-64): sólo para identificadores de vectores.
- **Número de parámetros** de función.
- **Posición del parámetro** dentro de la lista de parámetros (0 a n-1).
- **Número de variables locales** dentro de una función (0 a n-1).
- **Posición de la variable local** dentro de una función (entre 1 a n).

Ámbitos de la TABLA DE SÍMBOLOS para ALFA

Código fuente ALFA

```
main {

  //Variables globales
  int g1, g2, …;

  //Definición de funciones
  function int f1(int p1, int p2, …) //Parámetros para f1
  {
    // Variables locales f1
    int l1, l2, …;
    ...
  }
  function int f2(int q1, int q2, …) //Parámetros para f2
  {
    // Variables locales f2
    int m1, m2, …;
    ...
  }

  // Sentencias del programa principal
  ...
}
```

Ámbitos de la TABLA DE SÍMBOLOS para ALFA

Código fuente ALFA

```
main {

  //Variables globales
  int g1, g2, …;

  //Definición de funciones
  function int f1(int p1, int p2, …) //Parámetros para f1
  {
    // Variables locales f1
    int l1, l2, …;
    ...
  }
  function int f2(int q1, int q2, …) //Parámetros para f2
  {
    // Variables locales f2
    int m1, m2, …;
    ...
  }

  // Sentencias del programa
  ...
}
```

Ámbito	Declara	Variables
Global	g1, g2, f1, f2	g1, g2, f1, f2
Función f1	p1, p2, l1, l2	g1, g2, f1, p1, p2, l1, l2
Función f2	q1, q2, m1, m2	g1, g2, f1, f2, q1, q2, m1, m2

Información de la TABLA DE SÍMBOLOS para ALFA

Fichero: tablaHash.h

```
/*************** CONSTANTES ***************/

/* posición inicial de parámetros de función (empiezan a contar en 0) */
#define POS_INI_PARAMS 0
/* posición de inicio de variables locales de función (empiezan a contar en 1) */
#define POS_INI_VARS_LOCALES 1

#define HASH_INI 5381
#define HASH_FACTOR 33

/*************** DECLARACIONES DE TIPOS ***************/

/* Retorno de función error/ok */
typedef enum { ERR = 0, OK = 1 } STATUS;

/* Categoría de un símbolo: variable, parámetro de función o función */
typedef enum { VARIABLE, PARAMETRO, FUNCION } CATEGORIA;

/* Tipo de un dato: sólo se trabajará con enteros y booleanos */
typedef enum { ENTERO, BOOLEANO } TIPO;

/* Clase de un símbolo: pueden ser variables atómicas (escalares) o listas/arrays
 (vectores) */
typedef enum { ESCALAR, VECTOR } CLASE;
```

Información de la TABLA DE SÍMBOLOS para ALFA

Fichero: tablaHash.h

```
/* Información de un símbolo */
typedef struct {
    char *lexema;           /* identificador */
    CATEGORIA categoria;    /* categoría */
    TIPO tipo;              /* tipo */
    CLASE clase;            /* clase */
    /* valor si escalar, longitud si vector, número de parámetros si función */
    int adicional1;
    /*  posición en llamada a función si parámetro, posición de declaración si
        variable local de función, número de variables locales si función */
    int adicional2;
} INFO_SIMBOLO;

/* Nodo de la tabla hash */
typedef struct nodo_hash {
    /* información del símbolo */
    INFO_SIMBOLO *info;
    /* puntero al siguiente nodo (encadenamiento si colisión en misma celda) */
    struct nodo_hash *siguiente;
} NODO_HASH;

/* Tabla hash */
typedef struct {
    int tam;            /* tamaño de la tabla hash */
    NODO_HASH **tabla;  /* tabla en sí (array de tam punteros a nodo) */
} TABLA_HASH;
```

Información de la TABLA DE SÍMBOLOS para ALFA

Fichero: tablaHash.h

```
/*************** FUNCIONES ***************/

INFO_SIMBOLO *crear_info_simbolo(const char *lexema, CATEGORIA categ, TIPO tipo,
CLASE clase, int adic1, int adic2);

void liberar_info_simbolo(INFO_SIMBOLO *is);

NODO_HASH *crear_nodo(INFO_SIMBOLO *is);

void liberar_nodo(NODO_HASH *nh);

TABLA_HASH *crear_tabla(int tam);

void liberar_tabla(TABLA_HASH *th);

unsigned long hash(const char *str);

INFO_SIMBOLO *buscar_simbolo(const TABLA_HASH *th, const char *lexema);

STATUS insertar_simbolo(TABLA_HASH *th, const char *lexema, CATEGORIA categ, TIPO
 tipo, CLASE clase, int adic1, int adic2);

void borrar_simbolo(TABLA_HASH *th, const char *lexema);
```

Programa de prueba para la TABLA DE SÍMBOLOS para ALFA

./pruebaTS <fichero_entrada> <fichero_salida>

Fichero de entrada

- **Inserción de un elemento** [N]≥0
  Definición> [ID] [N]
  Ejemplo> global 10
- **Búsqueda de un elemento**
  Definición> [ID]
  Ejemplo> global
- **Apertura de un ámbito** [N]≤-1
  Definición> [ID] [N]
  Ejemplo> función -20
- **Cierre de ámbito activo**
  Definición> cierre -999
  Ejemplo> cierre -999

Fichero de salida

- **Inserción/Apertura con con éxito**
  Definición> [ID]
  Ejemplo> global
- **Inserción/Apertura sin éxito**
  Definición> -1 [ID]
  Ejemplo> -1 nombre
- **Búsqueda con éxito**
  Definición> [ID] [N]
  Ejemplo> global 10
- **Búsqueda sin éxito**
  Definición> [ID] -1
  Ejemplo> nombre -1
- **Cierre de ámbito local**
  Definición> cierre
  Ejemplo> cierre

[ID]: *Identificador*
[N]: *Número entero*

T Programa de prueba para la TABLA DE SÍMBOLOS para ALFA

Fichero de entrada

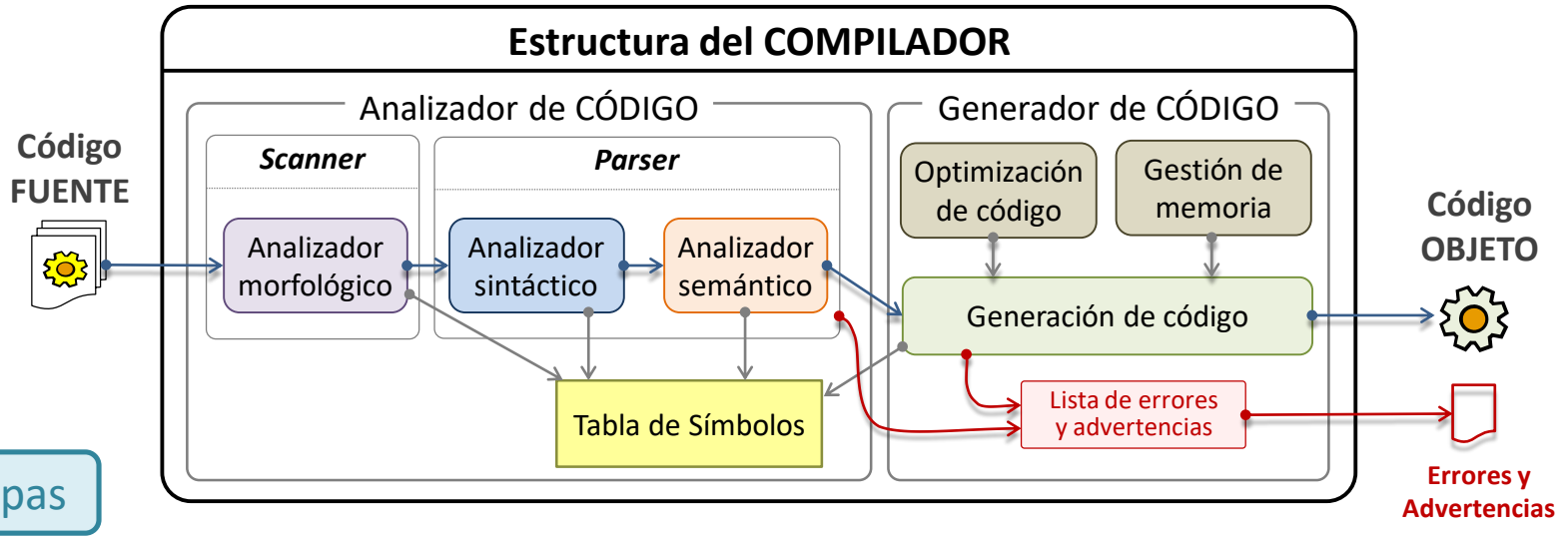
```
uno       1
dos       2
uno       10
dos       20
uno
dos
cuatro
cinco
funcion1 -10
uno       10
uno       20
funcion1 100
tres      3
funcion1
tres
dos
cuarto
cierre    -999
funcion1
```

Fichero de salida

```
uno
dos
-1        uno
-1        dos
uno       1
dos       2
cuatro    -1
cinco     -1
funcion1
uno
-1        uno
-1        funcion1
tres
funcion1 -10
tres      30
dos       2
cuarto    -1
cierre
funcion1 -10
```

Acción

```
Inserción con éxito.
Inserción con éxito.
Fallo de inserción.
Fallo de inserción.
Búsqueda con éxito.
Búsqueda con éxito.
Fallo de búsqueda.
Fallo de búsqueda.
Apertura de ámbito local.
Inserción con éxito.
Fallo de inserción.
Fallo de inserción.
Inserción con éxito.
Búsqueda con éxito.
Búsqueda con éxito.
Búsqueda con éxito.
Fallo de búsqueda.
Cierre de ámbito.
Búsqueda con éxito.
```



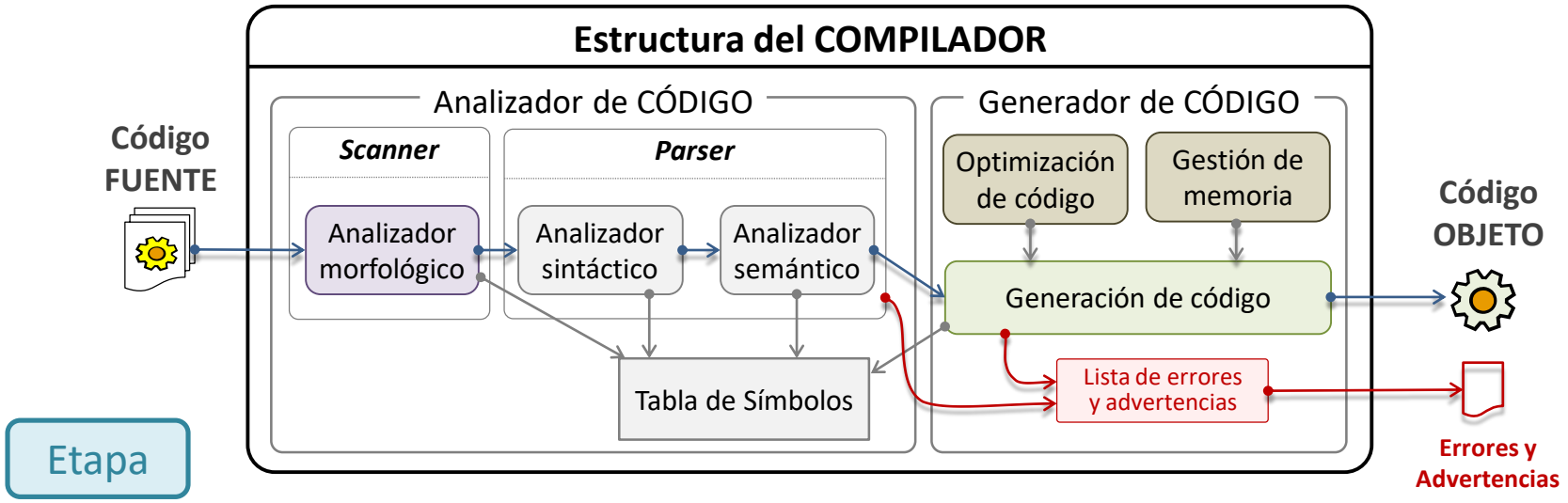
Etapas

1 **Analizador morfológico**: transforma el código fuente del programa de una secuencia de caracteres, a una secuencia de unidades sintácticas (tokens).

2 **Analizador sintáctico**: interpreta las unidades sintácticas identificadas (tokens) por el analizador morfológico como un programa estructurado según una gramática (ya se verá en Autómatas y Lenguajes).

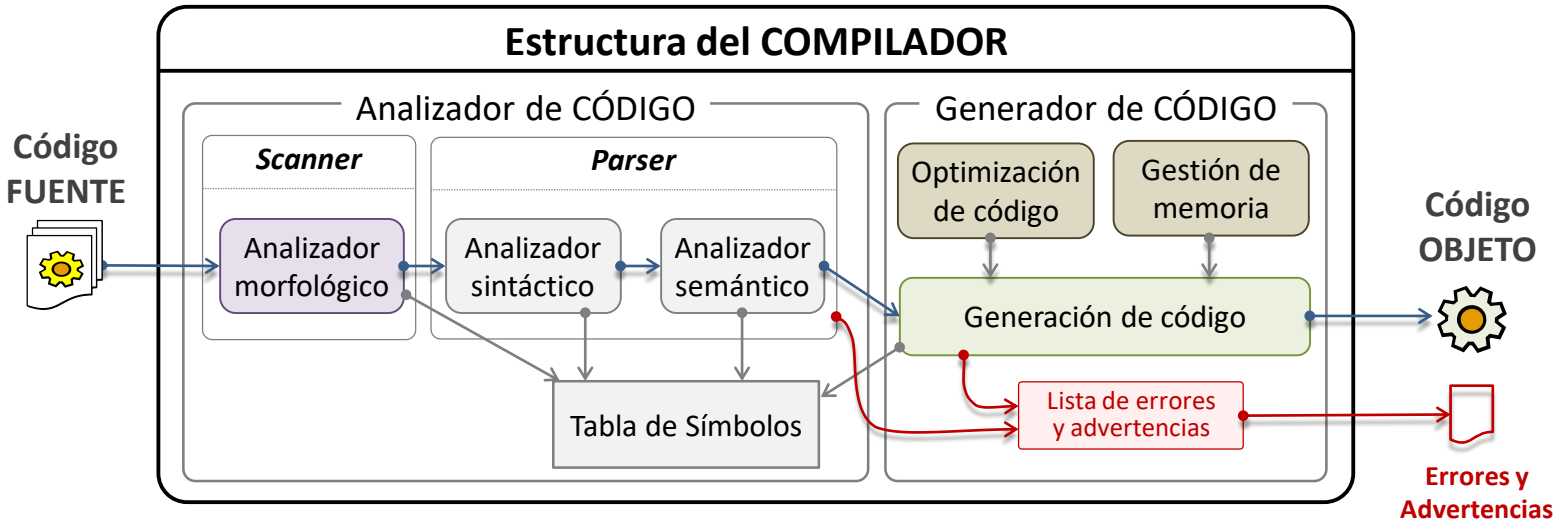
3 **Analizador semántico**: realiza comprobaciones semánticas como por ejemplo que las variables están declaradas antes de su uso, que los tipos de las expresiones son correctos, etc.

T **Tabla de símbolos o identificadores**. Se encarga de todos los aspectos dependientes del contexto relacionados con los *nombres* (variables, constantes, funciones, palabras reservadas, etc.) que puedan aparecer en los programas.



1 Analizador morfológico

- También se llama analizador léxico o scanner. Se hace cargo del análisis más sencillo. Elimina los comentarios.
- **Transforma** el código fuente en una secuencia de unidades sintácticas (tokens):
  - [ID] Identificadores
  - [PC] Palabras reservadas o palabras claves (begin, for, if, end,…)
  - [CN] Constantes numéricas (enteras, reales, etc...)
  - [CS] Constantes literales o de tipo “string”.
  - [SS] Símbolos simples (operadores, +, -, *,..., separadores, ;, ., ...)
  - [SM] Símbolos múltiples (operadores, +=, -=, >=, <=,...)
- Esta transformación permite al analizador sintáctico no vea el fichero como una secuencia de caracteres sino como una secuencia de elementos con significado sintáctico (tokens).



Código FUENTE

```
begin
  int A;
  A := 100;
  A := A+A;
  output A
end
```

1 Análisis morfológico

```
(<PC>,begin)
  (<PC>,int) (<ID>,A) (<SS>,;)
  (<ID>,A) (<SS>,:=) (<CN>,100) (<SS>,;)
  (<ID>,A) (<SS>,:=) (<ID>,A) (<SS>,+) (<ID>,A) (<SS>,;)
  (<PC>,output) (<ID>,A)
(<PC>,end)
```

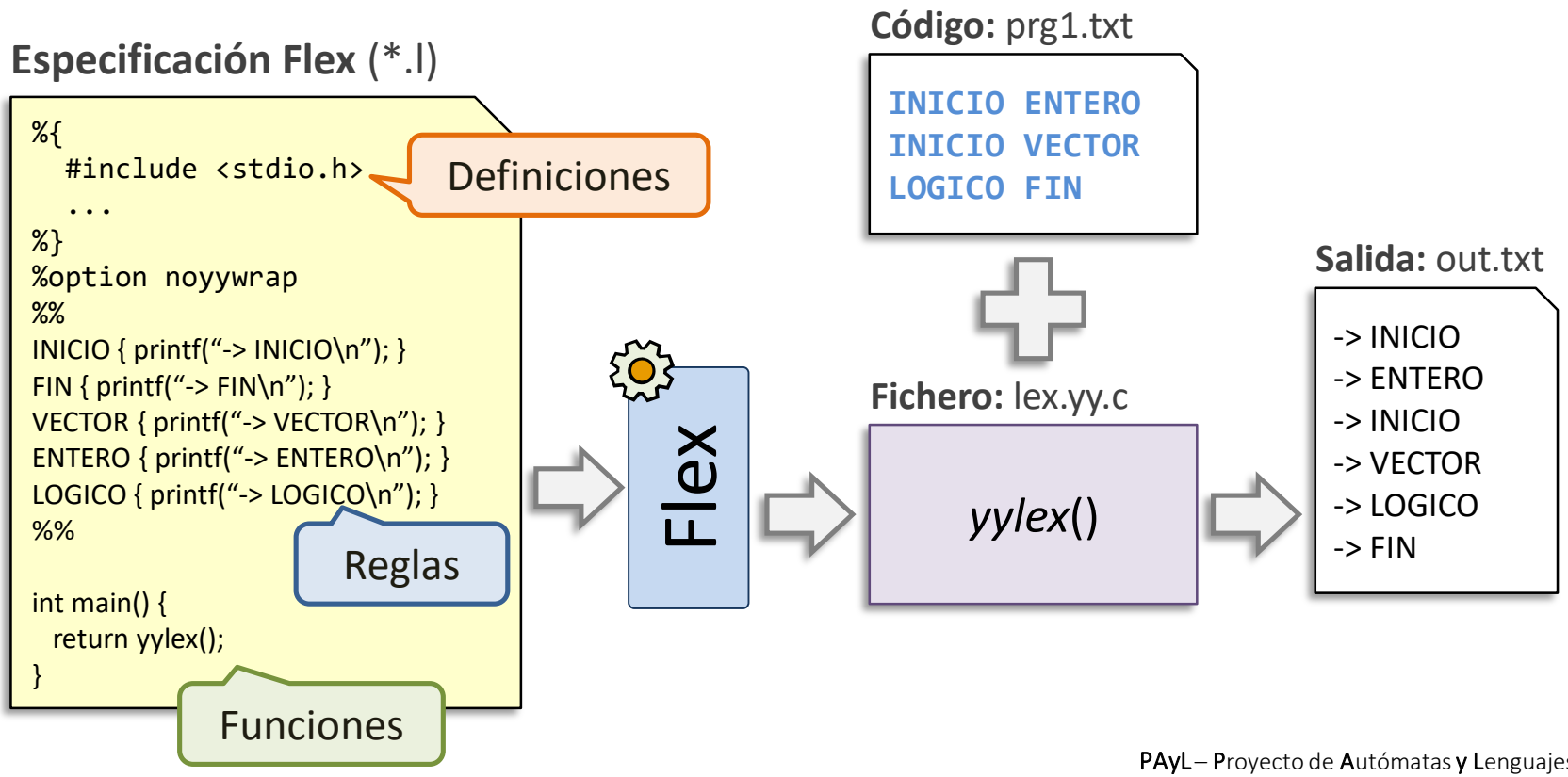
Token (<..>,valor)

Tokens

<ID> Identificadores, **<PC>** Palabras reservadas, **<CN>** Constantes numéricas; **<CS>** Constantes literales; **<SS>** Símbolos simples; **<SM>** Símbolos múltiples

1 Flex: herramienta para la creación del Análisis Morfológico

- **Flex** es una herramienta que permite **generar automáticamente autómatas finitos que reconocen lenguajes regulares expresados mediante patrones Flex**.
- Los **patrones** de Flex **son extensiones de las expresiones regulares**.
- Flex recibe como entrada **un lenguaje regular expresado como un conjunto de patrones**, y genera la función C ***yylex***() que es un autómata finito que **reconoce cadenas** pertenecientes a dicho lenguaje de entrada y opcionalmente **ejecuta alguna acción** cada vez que se reconoce una de las cadenas del lenguaje.
- A la entrada a la herramienta Flex se le llama **especificación Flex**.



A partir de la especificación <file>.l se genera el ejecutable de la siguiente manera:

0 Instalación Flex

```
$ sudo apt-get update
$ sudo apt-get install flex
```

1 Compilar la Especificación Flex

```
$ flex <file>.l
> lex.yy.c  //Genera el fichero
```

2 Generación de ejecutable

```
$ gcc –Wall –o execute lex.yy.c
```

3 Ejecución

```
$ ./execute
```