

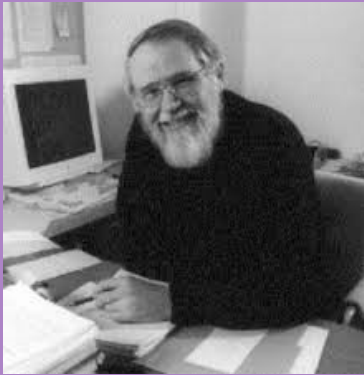
---

FEBRERO 2022

# COMPILADOR SENCILLO DE UNA PASADA

BLANCA ESTELA VILLALVAZO  
FLORES





BRIAN WILSON KERNIGHAN

?

---

## SINTAXIS DEL LENGUAJE

- Gramáticas independientes del contexto BNF (Manera formal de describir lenguajes formales).

For specifying SYNTAX, use a notation called "context-free grammar" or BNF  
↓  
Backus-Naur  
Form

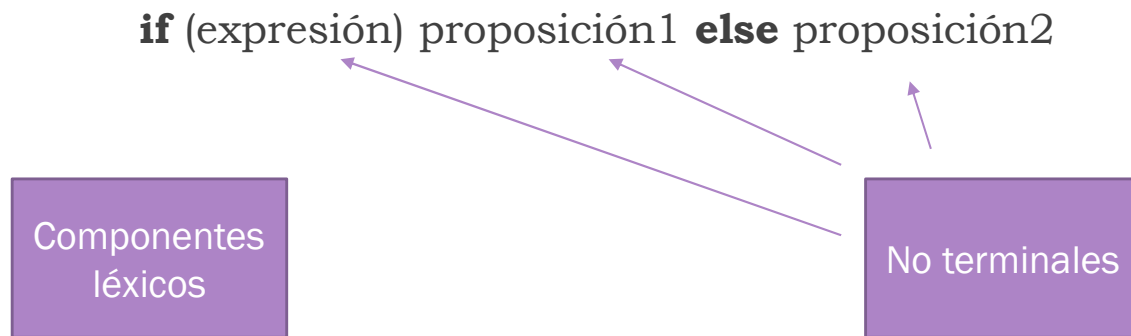
## SEMÁNTICA DEL LENGUAJE

- Orientación/Descripción mas informal

# Definición de la sintaxis

---

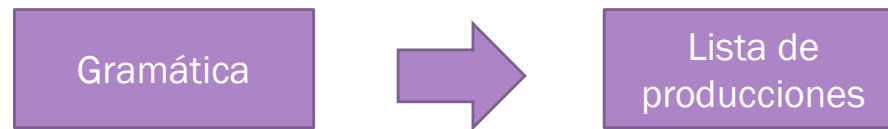
Una gramática describe de forma natural la estructura jerárquica de muchas construcciones de los lenguajes de programación.



# Gramática independiente del contexto, componentes:

---

1. Conjunto de componentes léxicos, denominados símbolos **terminales**.
2. Conjunto de **no** terminales.
3. Conjunto de **producciones**, cada producción consta de un no terminal, lado izquierdo, una flecha y una secuencia de componentes léxicos y/o no terminales, lado derecho.
4. La denominación de uno de los no terminales como símbolo inicial.



---

TERMINALES  
<= y cadenas en negritas  
(**while**)

NO TERMINALES  
cadenas en *cursivas*

COMPONENTE LEXICO  
Nombre o símbolo

# Ejemplo 1

Sea el caso de 9-5+2, 3-1 y 7, “Listas de dígitos separados por signos mas o menos”.

La siguiente gramática describe la sintáxis de esas expresiones. Las producciones son:

*lista* -> *lista* + *dígito*

*lista* -> *lista* – *dígito*

*lista* -> *dígito*

*dígito* -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

*lista* -> *lista* + *dígito* | *lista* – *dígito* | *dígito*


Componentes léxicos

+ - 0 1 2 3 4 5 6 7 8 9

*lista*

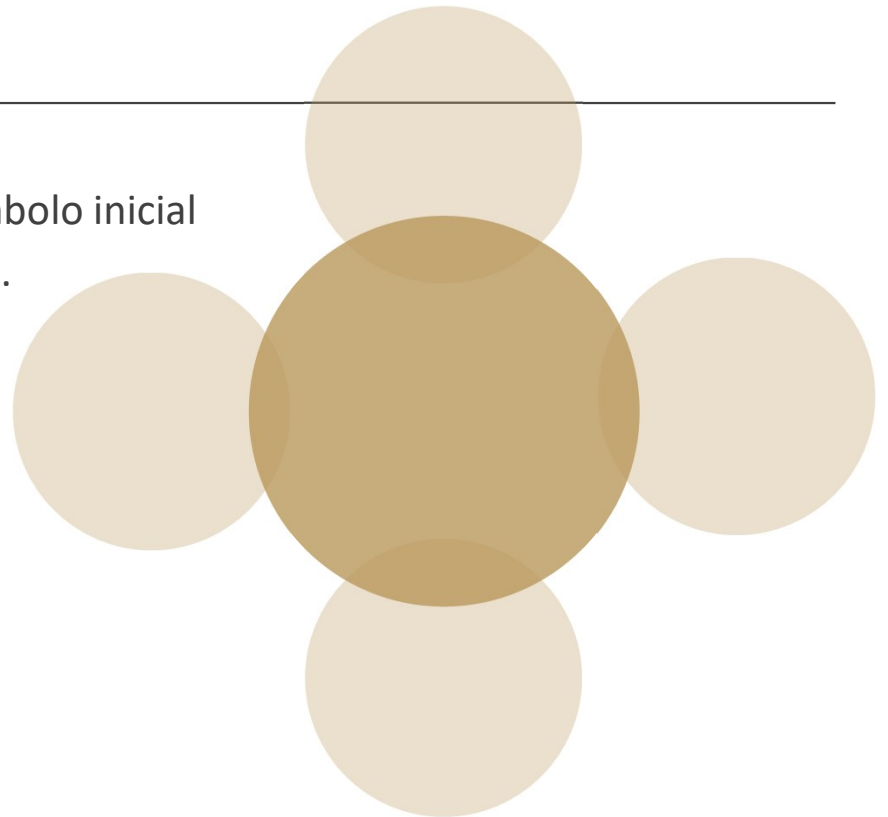
no terminal inicial

(sus producciones se dieron primero)



---

Las cadenas de componentes léxicos derivados del símbolo inicial  
forman el *lenguaje* que define la gramática.





## Ejemplo 1, continuación

Las diez producciones para el no terminal *dígito* hacen posible la representación de cualquiera de los componentes léxicos 0, 1, ...,9. Un dígito por sí solo es una lista.

9-5+2 es una *lista* por:

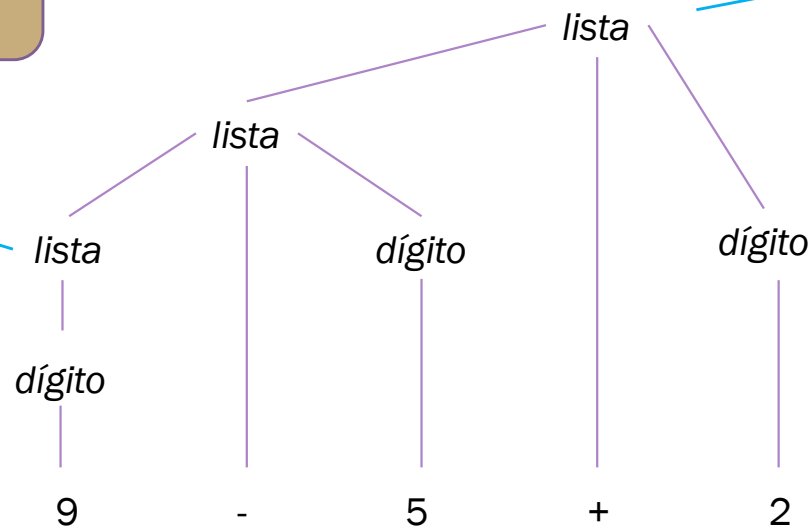
- a. 9 es una *lista* dado que 9 es un *dígito*.
- b. 9-5 es una *lista* dado que 9 es una lista y 5 es un *dígito*.
- c. 9-5+2 es una *lista* dado que 9-5 es una lista y 2 es un *dígito*.

## Ejemplo 1, continuación

Cómo es que el símbolo inicial de una gramática, deriva a una cadena del lenguaje

Árbol de análisis sintáctico para 9-5+2

No terminal



Símbolo inicial

Los hijos están etiquetados de izquierda a derecha

Componente léxico ó  $\epsilon$

## Ejemplo 2

Secuencia de proposiciones separadas por los símbolos de punto y coma que se encuentran en los bloques de **begin** y **end**.

Una gramática incluiría las producciones:

*bloque* -> **begin** *props\_opcional* **end**

*props\_opcional* -> *lista\_props* |  $\epsilon$

*lista\_props* -> *lista\_props* ; *prop* | *prop*

*prop* representarían proposiciones como if, asignaciones etc. (Diapositiva 17)

# Análisis sintáctico

---

Proceso de búsqueda de un árbol de análisis sintáctico para una cadena dada de componentes léxicos.

Lenguaje generado por una gramática:  
Es el conjunto de cadenas que pueden ser generadas por un árbol de análisis sintáctico.

# Ambigüedad

---

Una gramática puede tener **más** de **un** árbol de análisis sintáctico.

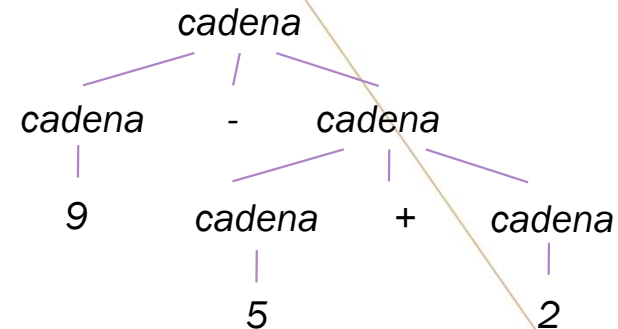
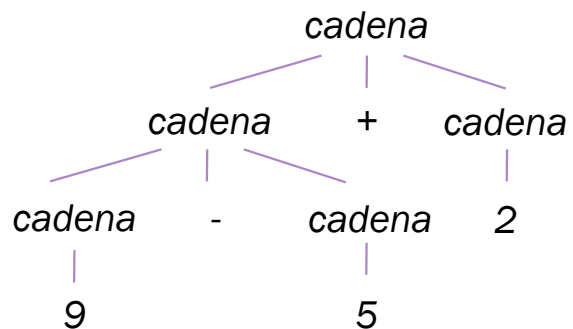
Para demostrar que una gramática es ambigua, lo único que se requiere es encontrar una cadena de componentes léxicos que tenga más de un árbol de análisis sintáctico.

## Ejemplo 3 (Basado en ejemplo 1)

Supóngase que no se hizo la distinción entre dígitos y listas.

*cadena*  $\rightarrow$  *cadena* + *cadena* | *cadena* - *cadena* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

9-5+2, se reagrupa en: (9-5)+2 ó 9-(5+2)



# Asociatividad de operadores

---

Asociativos por la izquierda : operadores aritméticos (+ - \* /)

Asociativo por la derecha : exponenciación, asignación.

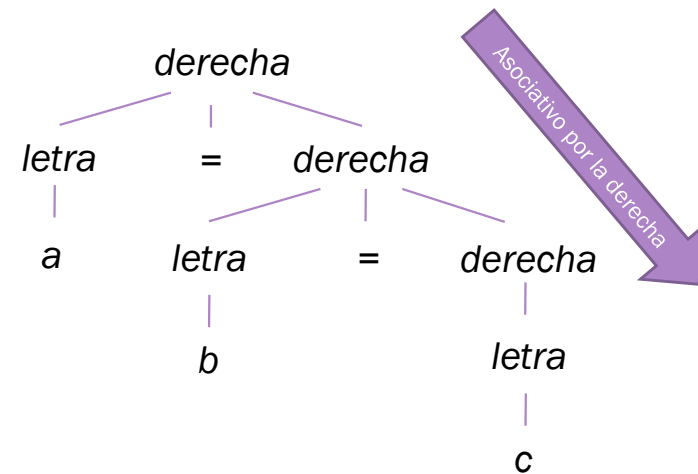
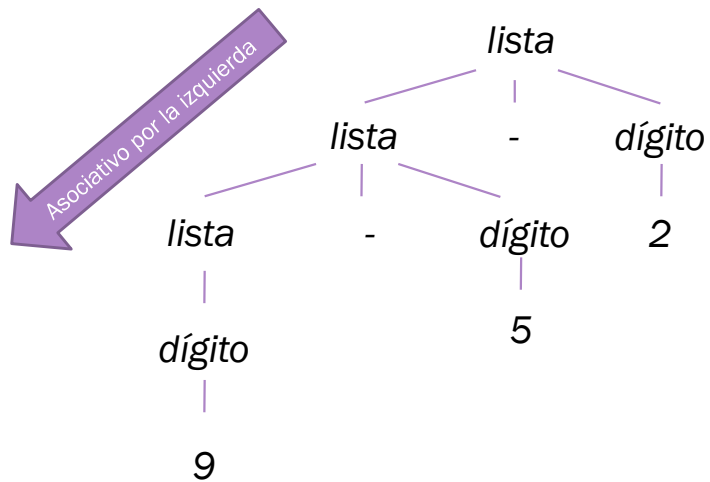
*derecha -> letra = derecha | letra*

*letra -> a | b | ... | z*

$a=b=c$   
Es lo mismo  
que  
 $a=(b=c)$

# Asociatividad de operadores

---





# Precedencia de operadores

---

$$9+5*2$$

$$(9+5)*2 \text{ o } 9+(5*2) \text{ ?}$$

Tarea:

- a) Construir la gramática para expresiones aritméticas (suma, resta, multiplicación y división)
- b) Construir la gramática para las proposiciones faltantes del ejemplo 2, [prop.](#)

# Traducción dirigida por la sintaxis

---

Notación postfija

$$(9-5)+2 \quad \longrightarrow \quad 95-2+$$

$$9-(5+2) \quad \longrightarrow \quad 952+-$$

# Definiciones dirigidas por la sintáxis

---

Para cada entrada  $x$ :

1.-Construir un árbol de análisis sintáctico.



El valor  $X.a$  en  $n$  se calcula por la regla semántica para el atributo  $a$  asociado con la producción de  $X$  utilizada en el nodo  $n$ .

# Emisión de una traducción

---

PRODUCCIÓN

$expr \rightarrow expr_1 + \text{término}$

$resto \rightarrow \text{término} + resto_1$

REGLA SEMÁNTICA

$expr.t := expr_1.t \ || \ \text{término}.t \ || \ '+'$

$resto.t := \text{término}.t \ || \ '+' \ || \ resto_1.t$

Concatenación



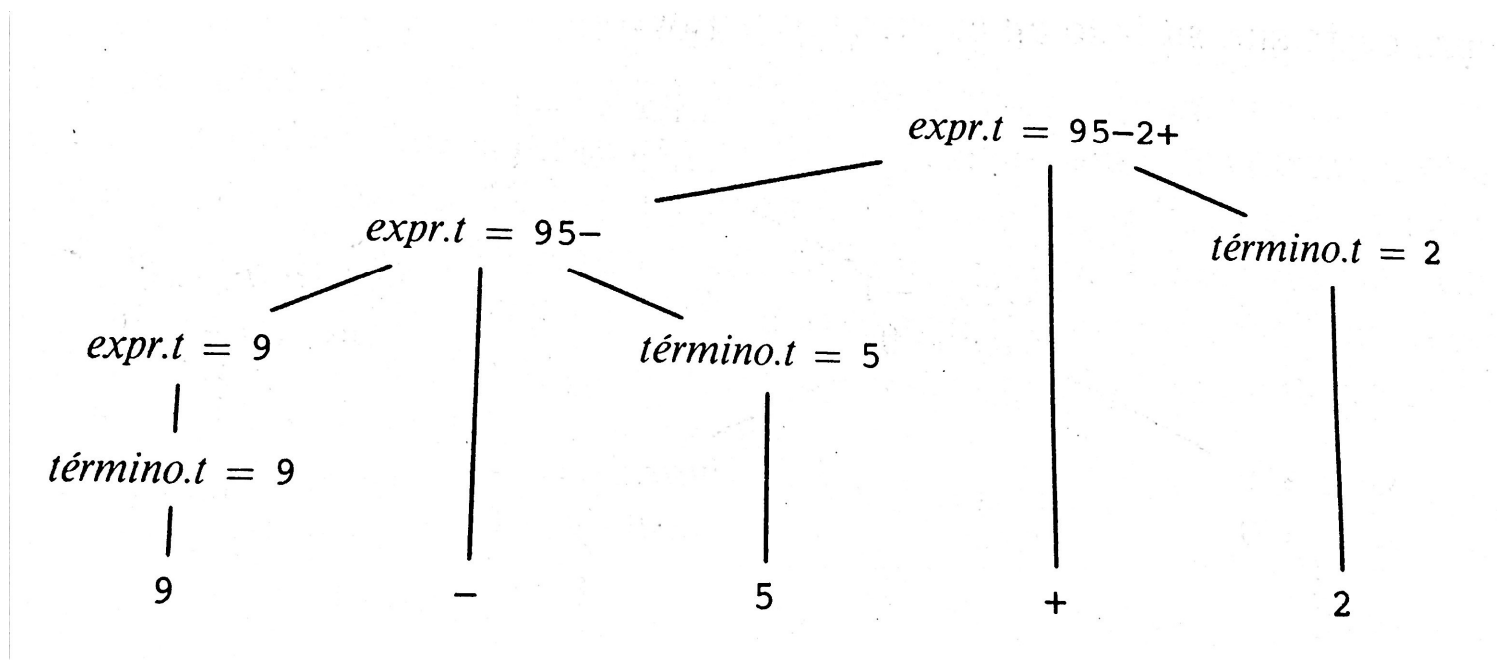
*Imprimen cadenas como:*

$expr \rightarrow expr_1 + \text{término} \{ \text{print} ('+') \}$

$resto \rightarrow + \text{término} \{ \text{print} ('+') \} resto_1$

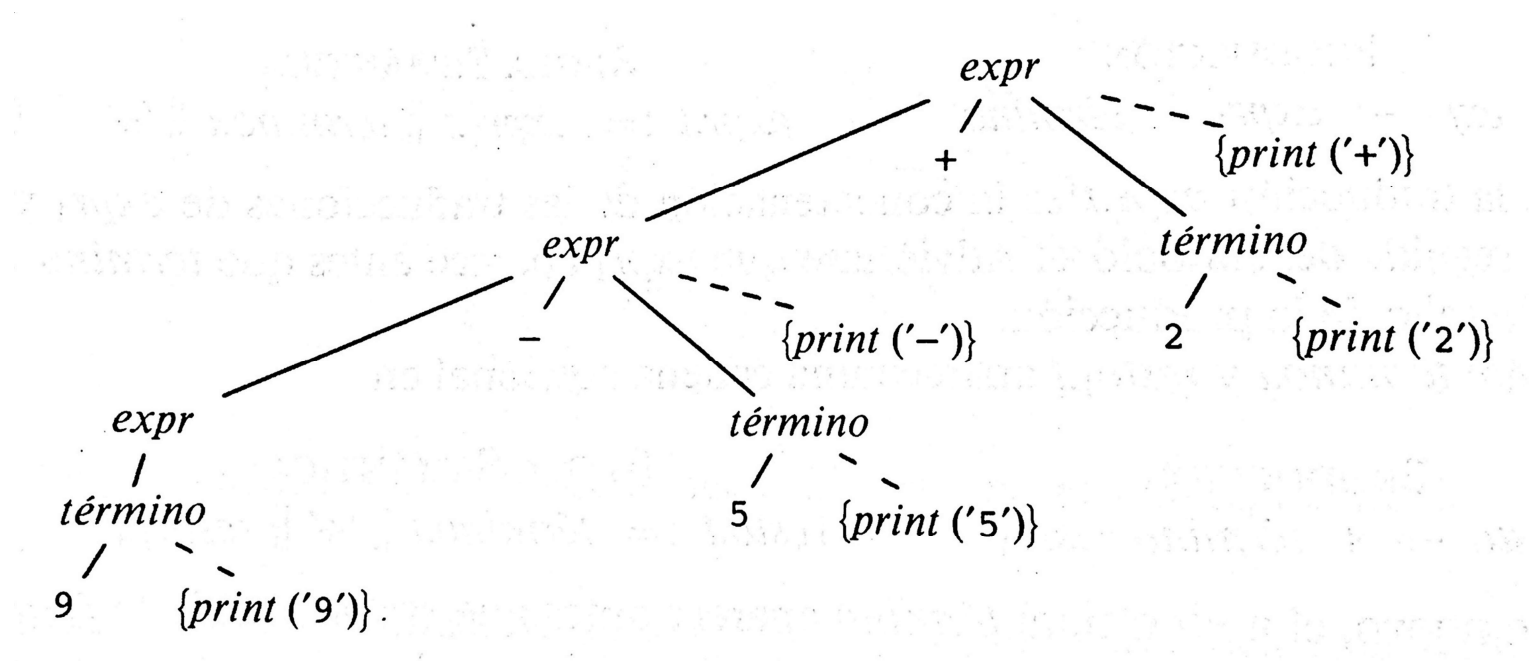
# Valores de atributos en los nodos de un árbol de análisis sintáctico

---



# Árbol de análisis sintáctico con acciones para 9-5+2

---





---

### Tarea:

Investigar lo que es un algoritmo voraz y sus aplicaciones. Elaborar el reporte correspondiente.

### Programa:

Elabore un programa que de entrada a una expresión aritmética infija y dé la salida en forma postorden y en profundidad. Elabore el reporte correspondiente que incluya

- a) Portada
- b) Programa comentado, explicando lo mas que se pueda cada línea o procedimiento.
- c) 3 ejemplos de entradas y sus correspondientes lecturas en formato postfijo y profundidad. Impresiones de pantalla.
- d) Conclusiones personales.

# Análisis sintáctico

---

Proceso de determinar si una cadena de componentes léxicos puede ser generada por una gramática.

Para cualquier gramática independiente del contexto hay un analizador sintáctico que toma como máximo un tiempo de  $O(n^3)$  para hacer el análisis de una cadena de  $n$  componentes léxicos.



Hacen referencia al orden en que se construyen los nodos de un árbol de análisis sintáctico.



# Programa

Compila el siguiente programa y responde las siguientes preguntas:

- 1.- ¿Cuándo se ejecuta el método `error()`?
- 2.- ¿Para qué necesita un parámetro (`t`) el método `parea`?
- 3.- ¿Cuántos caracteres puede recibir a la entrada?
- 4.- ¿Cuál es el objetivo de añadir un salto de línea en el `main` después de evaluar `expr()`?
- 5.- ¿Para qué sirve el método `parea(x)`?

Elabora el reporte correspondiente y subelo al moodle.

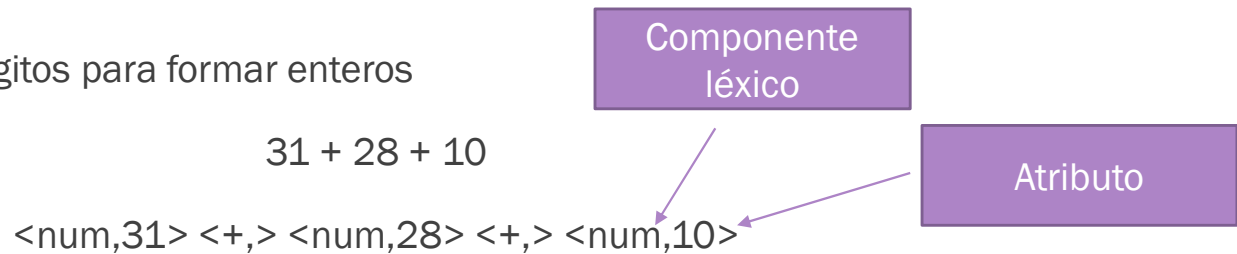
```
#include <ctype.h> /* carga el archivo que contiene al
                    predicado isdigit */
int preanálisis;
main()
{
    preanálisis = getchar();
    expr();
    putchar('\n'); /* agrega un carácter de línea nueva al final */
}
expr()
{
    término()
    while(1)
        if (preanálisis == '+') {
            parea('+'); término(); putchar('+');
        }
        else if (preanálisis == '-') {
            parea('-'); término(); putchar('-');
        }
        else break;
}
término()
{
    if (isdigit(preanálisis)) {
        putchar(preanálisis);
        parea(preanálisis);
    }
    else error();
}
parea(t)
int t;
{
    if (preanálisis == t)
        preanálisis = getchar();
    else error();
}
error()
{
    printf("error de sintaxis\n"); /* imprime mensaje de error */
    exit(1);                      /* y después se detiene */
}
```

## ...tener en mente

---

Eliminación de los espacios en blanco, si los elimina el analizador léxico, el analizador sintáctico tiene menos consideraciones.

Constantes, agrupación de dígitos para formar enteros



Reconocimiento de identificadores y palabras clave, usan la tabla de símbolos para revisar su aparición

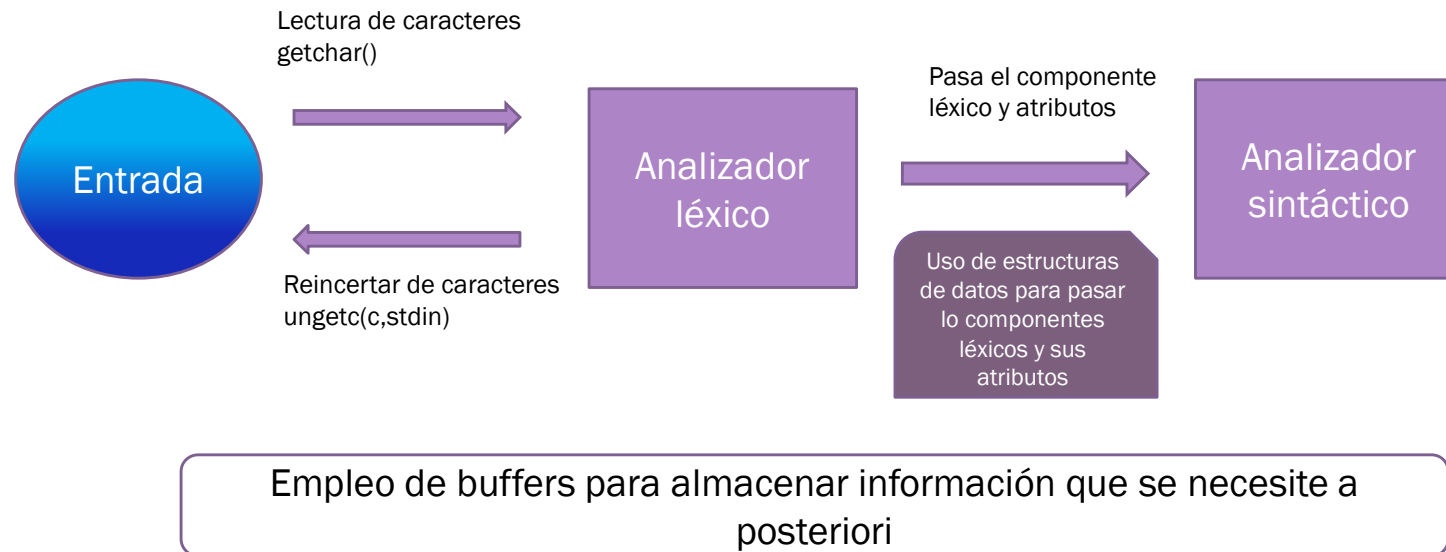
ejemplo = ejemplo + incremento;

id = id + id;

Identificador  $\neq$  palabra reservada

# Inserción del analizador léxico entre la entrada y el analizador sintáctico

---



# Anexo de números

---

Reemplazar los dígitos por:

factor -> (expr) | num {print{num.valor}}

```
factor()
{
    if (preanálisis == '(') {
        parea('('); expr(); parea(')');
    }
    else if (preanálisis == NUM) {
        printf(" %d ", valcomplex); parea(NUM);
    }
    else error();
}
```

# Analizador léxico que elimina espacios en blanco y reconoce números

```
(1) #include <stdio.h>
(2) #include <ctype.h>
(3) int numlínea = 1;
(4) int valcomplex = NINGUNO;

(5) int análex()
(6) {
(7)     int t;
(8)     while(1) {
(9)         t = getchar();
(10)        if (t == ' ' || t == '\t')
(11)            ; /* elimina espacios en blanco y símbolos tab */
(12)        else if (t == '\n')
(13)            numlínea = numlínea + 1;
(14)        else if (isdigit(t)) {
(15)            valcomplex = t - '0';
(16)            t = getchar ();
(17)            while (isdigit(t)) {
(18)                valcomplex = valcomplex*10 + t-'0';
(19)                t = getchar();
(20)            }
(21)            ungetc(t, stdin);
(22)            return NUM;
(23)        }
(24)        else {
(25)            valcomplex = NINGUNO;
(26)            return t;
(27)        }
(28)    }
(29) }
```

# Tabla de símbolos

---

Interfaz de la tabla de símbolos

**inserta(s,t):** devuelve el índice de la entrada.  
s: cadena  
t: componente léxico

**busca(s):** devuelve el índice de la entrada para la cadena s | 0

Palabras reservadas

inserta("div", div);

Inserta("mod",mod);

# Máquinas de pilas abstractas

---

## INSTRUCCIONES ARITMÉTICAS

Evalúe la expresión postfija:  $1\ 4\ +\ 2\ *$

## VALORES DEL LADO IZQUIERDO Y DERECHO

$a = 2;$  punteros vs posiciones

$a = a + 4;$

# Máquinas de pilas abstractas

---

## TRADUCCIÓN DE ASIGNACIONES A CÓDIGO MÁQUINA DE LA PILA

día = (1420\*a) div 4 + (153\*m +2) div 5 + d

```
valori día
mete 1420
valord a
*
mete 4
div
mete 153
valord m
*
mete 2
+
mete 5
div
+
valord d
+
=
```



# Máquinas de pilas abstractas

---

## TRADUCCIÓN DE PROPOSICIONES

Se centra en la creación de etiquetas en la resolución de expresiones (if, while)

## EMISIÓN DE UNA TRADUCCIÓN

if t=blanco or t=tab then...

expr<sub>1</sub> or expr<sub>2</sub>

if expr<sub>1</sub> then true else expr<sub>2</sub>

---

Elabore el traductor con ayuda del material proporcionado.



# Módulo de análisis léxico

---

LEXEMA	COMPONENTE LÉXICO	VALOR DEL ATRIBUTO
Espacio en blanco		
Secuencia de dígitos	NUM	Valor numérico de la secuencia
Div /	DIV	
MOD	MOD	
Secuencia de caracteres, iniciando con letra	ID	Indice en la tabla de símbolos
Carácter fin	FIN	
Cualquier carácter	Ese carácter	.

---

busca(s)

Deberá determinar si ya existe el lexema en la tabla de símbolos.

Inserta(s,t)

Insertar el lexema nuevo en la tabla de símbolos. Devuelve el índice de la tabla de símbolos.

numLinea

Contador que incrementa cada vez que se encuentre una nueva línea.

Almacenamiento de palabras reservadas deberá ser similar a una tabla de símbolos.

# errores

---

Error de sintáxis

El compilador imprime un mensaje informando del error en la línea en curso.

Saltar al siguiente signo de punto y coma y continuar el análisis.