

## **Scanner para C – Análisis Léxico-Sintáctico**

Paola Chinchilla Sánchez, Fabrizio Ferreto Saborío y Carlos Varela Ramírez

Escuela de Computación, Tecnológico de Costa Rica

IC5701: Compiladores e Intérpretes

Ing. Erika Marín Schumann

Junio, 2021

## Contenido

Introducción .....	3
Estrategia de Solución.....	4
Análisis de Resultados .....	5
Lecciones Aprendidas .....	7
Casos de Prueba .....	8
Caso #1: Detección de errores.....	8
Caso #2: Errores sintacticos .....	9
Caso #3: Programa correcto .....	11
Manual de Usuario.....	12
Bitácora .....	13

## **Introducción**

Una de las partes más importantes en el proceso de compilación de los lenguajes de programación es el análisis sintáctico, pues es en esta fase donde toda la estructura de los programas es revisada y se asegura que todas las instrucciones tengan sentido, al menos estructuralmente.

Este proyecto tiene como objetivo el desarrollo de un programa capaz de analizar la estructura sintáctica de un programa y reportar los errores que se presenten en este de ser necesario.

## **Estrategia de Solución**

Para este proyecto era importante trabajar a partir del proyecto pasado. Lo primero fue analizar los documentos necesarios para el funcionamiento de la librería cup. Seguidamente fue necesario analizar el funcionamiento de dichos archivos y su compilación para comenzar su uso con Java. Luego tuvimos que comenzar la edición del archivo sintáctico. Para lograr esto la estrategia usada se basó en, primero desglosar las reglas sintácticas de C. Hecho esto comenzamos a agregar producciones una a una comprobando su compilación. Por último, agregamos producciones de error para comprobar los errores sintácticos presentes con las cuales se obtuvieron los errores producidos y se desglosaron en interfaz gráfica.

## Análisis de Resultados

Actividades	Estado
Listado de errores léxicos encontrados	<b>Completado al 100%</b> Se listan todos los errores léxicos encontrados.
Listado de errores sintácticos encontrados	<b>Completado al 90%.</b> Algunas expresiones complejas provocan que se salten errores.
Palabras reservadas	<b>Completado al 100%</b> Se implementaron todas las palabras reservadas solicitadas. Las palabras como continue o break pueden estar en cualquier parte del programa.
Funciones Read y Write	<b>Completado al 100%</b> Se implementaron las 2 nuevas funciones con sus condiciones especiales de parámetros.
Variables: char, int, long, short.	<b>Completado al 100%</b> Se implementaron todos los tipos, inclusive los diferentes tipos de long y shorts sumados a los tipos int.
Estructura del programa	<b>Completado al 95%</b> Identifica correctamente estructuras de un programa, sin embargo, cuando hay llaves faltantes en expresiones como if, puede que salte líneas a la hora de la recuperación

Estructura de las funciones	<b>Completado al 100%</b> La estructura de las funciones funciona correctamente, primero declaración de variables y luego el cuerpo de la función.
Operadores aritméticos y booleanos	<b>Completado al 100%</b> Se implementaron todos los operadores aritméticos (incluyendo unarios) y los booleanos de manera correcta.
En el cuerpo de una función puede venir expresiones, o estructuras de control, instrucciones como read, write, break, continue	<b>Completado al 100%</b> Dentro de una función se puede incluir cualquier tipo de sentencia menos definición de funciones.
Estructuras de control	<b>Completado al 100%</b> Se implementaron las estructuras if, if-else, for, while y switch. En cada uno de ellos se logra valorar errores en sus definiciones.
Las expresiones pueden ser asignaciones, expresiones aritméticas o booleanas o literales.	<b>Completado al 100%</b> Las expresiones pueden ser de cualquiera de estos tipos o combinaciones.
Mensajes de error	<b>Completado al 80%</b> Los mensajes de error muestran línea y columna, sin embargo, se dificultó la especificación de los errores según su tipo

## **Lecciones Aprendidas**

Este proyecto fue indudablemente retador debido a la cantidad de aspectos a tomar en cuenta en la sintaxis de C. Debido a esto también se aprendieron lecciones valiosas a la hora de programar. La más importante fue entender la importancia del orden y la organización con el fin de obtener el resultado deseado. Esto se debe a que, con la cantidad de producciones manejadas, el no haber mantenido estas características hubiera generado gran confusión a la hora de identificar errores y entender la rama de parseo seguida por el programa. Por otro lado, también aprendimos como el entendimiento de las herramientas usadas simplificaban el desarrollo de manera monumental. Por último, notamos como un buen trabajo en equipo generaba una mayor abundancia de buenas ideas que indudablemente ayudaron al resultado final del proyecto.

## Casos de Prueba

### Caso #1: Detección de errores

El programa debe ser capaz de reconocer errores sintácticos y léxicos y presentarle al usuario el listado en las tablas correspondientes.

#### Código de ejemplo:

```
int a;
char c;
int p, f;
void funcion(char x,long y){
long f;
a = 3 //error sintáctico
if (x>7){
p=7+ñ; //error léxico
}
return;
}
```

#### Resultados de prueba:

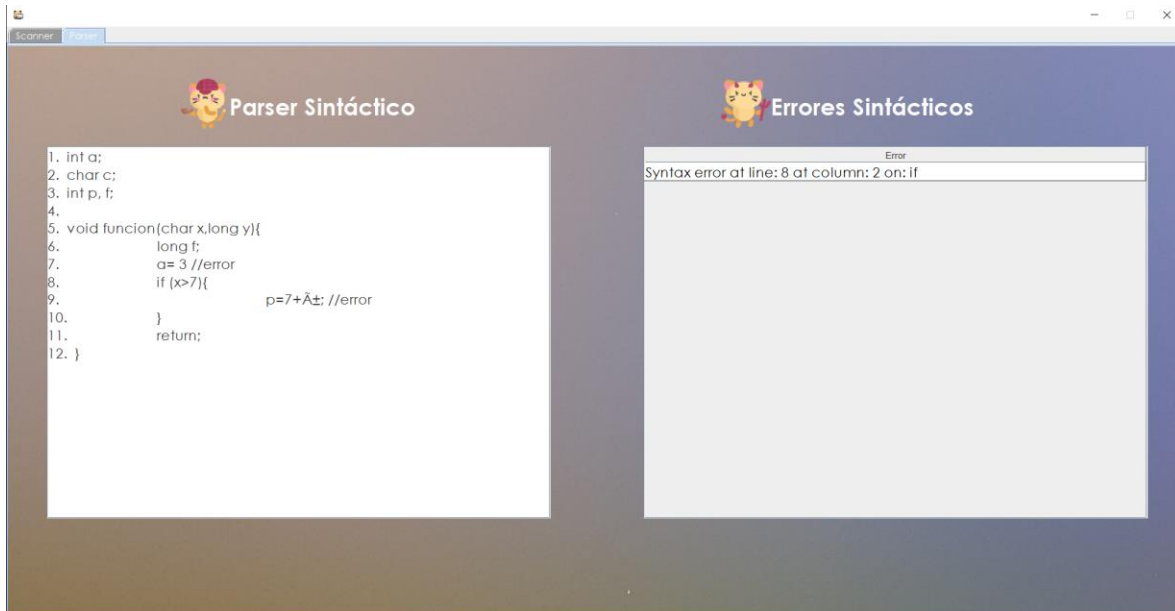
The screenshot displays a web-based application interface for a lexical scanner. It features two main panels: 'Scanner Léxico' on the left and 'Errores Léxicos' on the right. The 'Scanner Léxico' panel contains a table with three columns: 'Token', 'Tipo', and 'Lineas'. The 'Errores Léxicos' panel contains a table with two columns: 'Error' and 'Linea'. At the bottom right, there is a 'Cargar archivo' button and a small cat icon.

Token	Tipo	Lineas
elseelse	Identificador	13, 14
while	Reservadas	9, 10, 12, 15
long	Reservadas	19, 4, 5
funcion	Identificador	4
if	Reservadas	5, 7, 10, 11, 13
++	Operador	16, 17, 9, 11
==	Operador	10
a	Identificador	0, 17, 18, 21, 6, 9, 11, 12...
!	Operador	10, 11, 13
13	Literal	6
b	Identificador	18, 11, 12, 14
c	Identificador	1, 18, 10, 11, 13
void	Reservadas	4
funcion2	Identificador	12
f	Identificador	2, 19, 5, 14, 15
{	Operador	4, 5, 7, 9, 10, 11, 12, 13, ...
}	Operador	4, 5, 7, 9, 10, 11, 12, 13, ...
+	Operador	17, 18, 6, 8, 11, 12, 14

Error	Linea
±	6, 8
Ã	6, 8

Cargar archivo





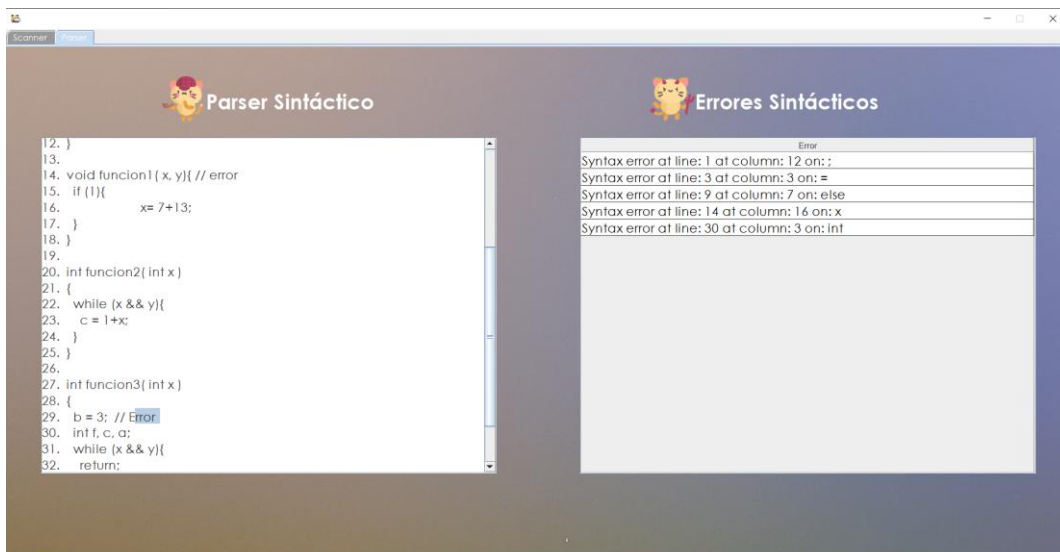
## Caso #2: Errores sintacticos

El programa debe ser capaz de retornar en la tabla “Errores Sintácticos” mensajes de error cuando se presentan palabras como reservadas cuando no lo son, errores con los operadores aritmeticos o booleanos, errores en las expresiones y en la estructura del programa o las funciones.

### Código de prueba:

```
char b, c f; //Error
int f, c, a;
b = 3; // Error
void funcion(int x,int y){
    if (y>=0) {
        x= a+b;
    } else else { //error
        return;
    } }
void funcion1( x, y){ // error
    if (1){
        x= 7+13;
    }
}
int funcion2( int x )
{
    while (x && y){
        c = 1+x; }
}
int funcion3( int x )
{
    b = 3; // Error
    int f, c, a;
    while (x && y){
        return; }
}
```

### Resultados de prueba:



The screenshot shows a web application interface for a syntactic parser. It is divided into two main panels: "Parser Sintáctico" on the left and "Errores Sintácticos" on the right. The left panel displays a list of 32 lines of C code. The right panel shows a table of syntax errors.

**Parser Sintáctico**

```
12. }
13.
14. void funcion1( x, y){ // error
15.     if (1){
16.         x= 7+13;
17.     }
18. }
19.
20. int funcion2( int x )
21. {
22.     while (x && y){
23.         c = 1+x;
24.     }
25. }
26.
27. int funcion3( int x )
28. {
29.     b = 3; // Error
30.     int f, c, a;
31.     while (x && y){
32.         return;
```

**Errores Sintácticos**

Error
Syntax error at line: 1 at column: 12 on: ;
Syntax error at line: 3 at column: 3 on: =
Syntax error at line: 9 at column: 7 on: else
Syntax error at line: 14 at column: 16 on: x
Syntax error at line: 30 at column: 3 on: int

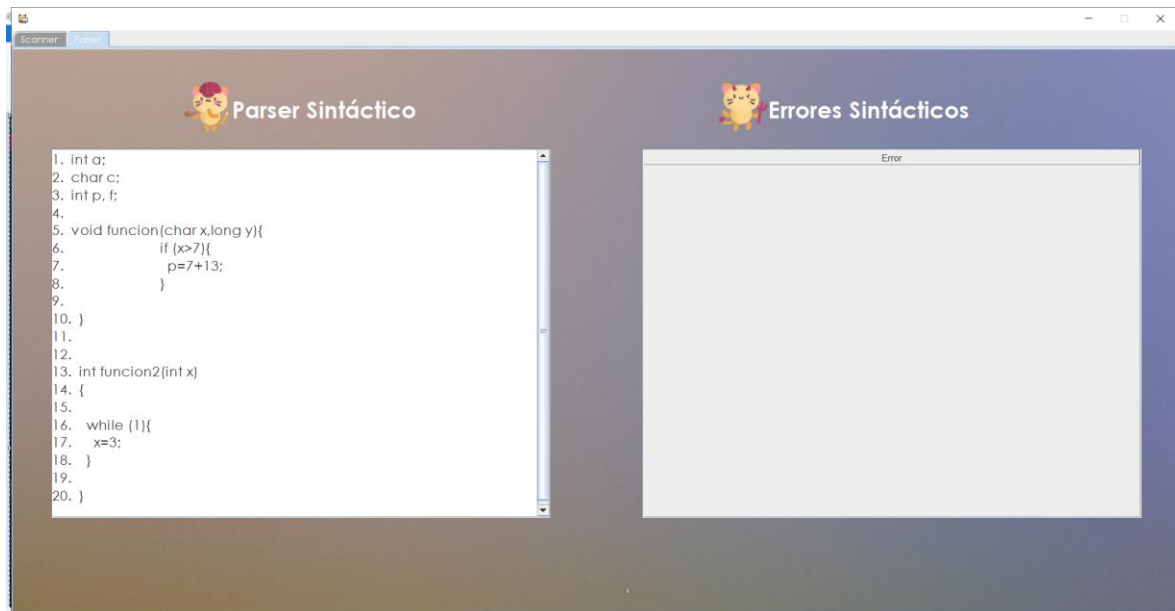
### Caso #3: Programa correcto

El programa debe ser capaz de retornar la tabla “Errores Sintácticos” sin ningún error cuando el código presentado posee una sintaxis correcta, no debe mostrar errores sin motivo aparente.

#### *Código de prueba:*

```
int a;  
char c;  
int p, f;  
void funcion(char x, long y){  
    if (x>7) {  
        p=7+13;  
    }  
}  
int funcion2(int x){  
    while(1){  
        x=3;  
    }  
}
```

#### *Resultados de prueba:*



## **Manual de Usuario**

Se puede acceder al manual de usuario del presente proyecto haciendo clic en este [hipervínculo](#).

## **Bitácora**

17 de mayo: Realizamos una reunión para un mejor entendimiento del proyecto.

20 de mayo: Discutimos lo aprendido sobre la librería Cup y como se debería llevar a cabo el desarrollo del proyecto.

22 de mayo: Iniciamos el uso de la librería cup para entender su uso en unión con la librería Flex.

23 de mayo: Comenzamos a arreglar el archivo .flex que utilizaría Cup para iniciar pruebas funcionales

26 de mayo: Iniciamos el archivo .cup que contiene la sintaxis para su futura compilación

28 de mayo: Comprendimos la modalidad de compilación de la librería cup y logramos la generación de los archivos “sym” y “parser”.

30 de mayo: Iniciamos la agregación de producciones al archivo de sintaxis y la interfaz gráfica.

31 de mayo: Continuamos agregando reglas al archivo de sintaxis y se finalizó la interfaz.

1 de junio: Arreglamos algunos errores de compilación provocados por las nuevas reglas de sintaxis y comenzamos la detección de errores.

2 de junio: Seguimos con la detección de los errores de Sintaxis.

3 de junio: Casi finalizamos la detección de errores de Sintaxis a excepción de pequeños problemas.

4 de junio: Se terminaron de arreglar los errores.

## **Bibliografía**

Louden, K. (2004). Construcción de compiladores (1st ed.). Thomson.

Hudson, S. (2021). CUP. Retrieved 5 June 2021, from <http://www2.cs.tum.edu/projects/cup/>

Charlead. (2019). JCup y JFlex | Analizador sintáctico con Java (explicación paso a paso) [Video]. <https://www.youtube.com/watch?v=4Z6Tnit810Y&t=396s>.

D.K Al. (2018). Installing and Configuring CUP Parser Generator [Video]. Retrieved 2021, from <https://www.youtube.com/watch?v=zWoDiDy5c-U>.