Avance 0 & 1:

0 - Tokens, Diagramas de Sintaxis y Gramática
1 - Sintaxis y Lexico

Propuesta

Durant

Tokens

En un compilador los tokens son un grupo de caracteres que juntos tienen un significado, normalmente las tokens son una palabra o signo de puntuación.

Estas tokens serán separadas por nuestro lexer y pasadas a nuestro parser para que se corran.

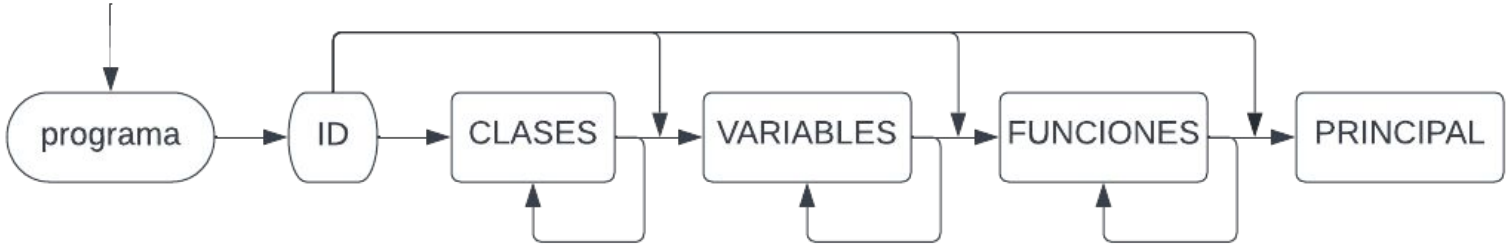
Palabras Clave	Identificadores	Literales	S. Puntuacion	Operadores
lee	int	cte-i	,	+
escribir	float	cte-f	;	-
if	char	cte-c	:	*
else	void		(	/
do	bool		)	=
while	Clase		[	<
for	atributos		]	>
hereda	metodos		{	<=
else if	principal		}	>=
	return			==
				!=
				||
				&&

Sintaxis

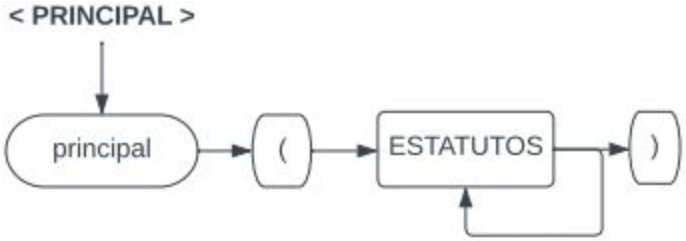
Una vez que tengamos las tokens declaradas nos toca trabajar con él sintaxis de nuestro compilador, es decir vamos a tomar los tokens ya leídos por el léxico y con esto formar el **parse tree.**

Vamos a buscar cualquier error que puede prevenir que se corra el programa.

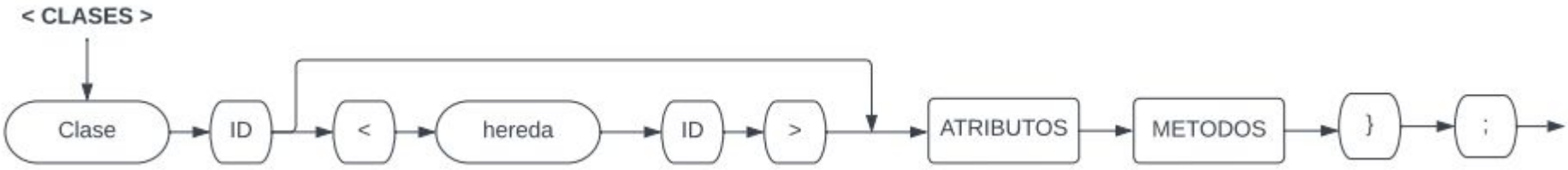
< PROGRAMA >



PROGRAMA → programa ID PROGRAMA2
PROGRAMA2 → CLASES | VARIABLES | FUNCIONES | PRINCIPAL
CLASES → CLASES | VARIABLES
VARIABLES → VARIABLES | FUNCIONES
FUNCIONES → FUNCIONES | PRINCIPAL

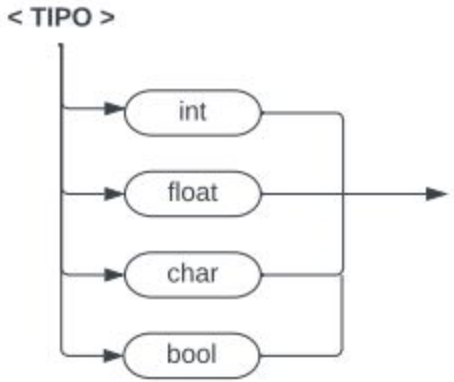


PRINCIPAL → principal (ESTATUTOS
ESTATUTOS → ESTATUTOS | ESTATUTOS2
ESTATUTOS2 →)

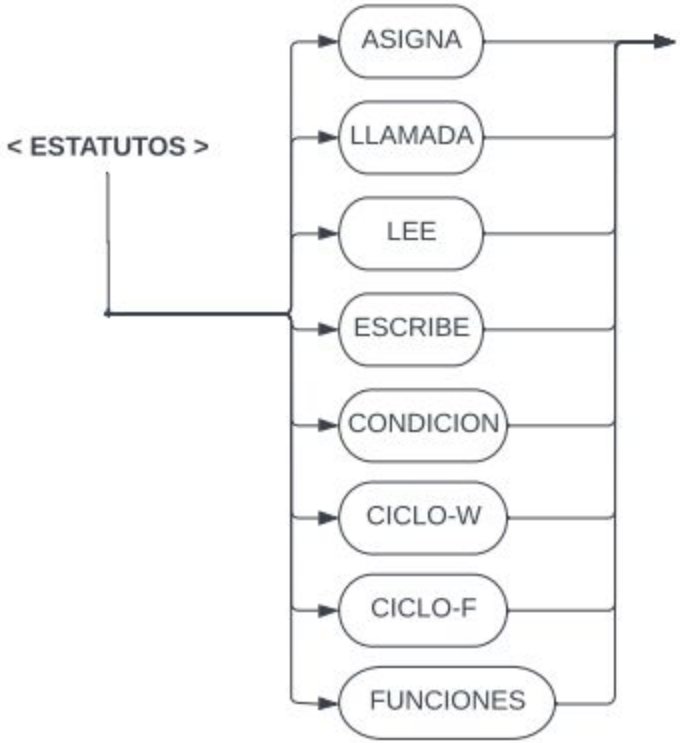


CLASES → Clase ID CLASES2
CLASES2 → CLASES3 ATRIBUTOS | ATRIBUTOS
CLASES3 → < hereda ID >
ATRIBUTOS → MÉTODOS) ;

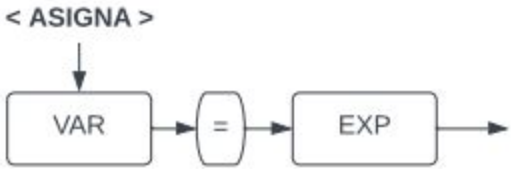
FUNCIONES → funcion FUNCIONES2
FUNCIONES2 → TIPO ID FUNCION3 | void ID FUNCION3
FUNCION3 → (PARAMETROS) { ESTATUTOS }



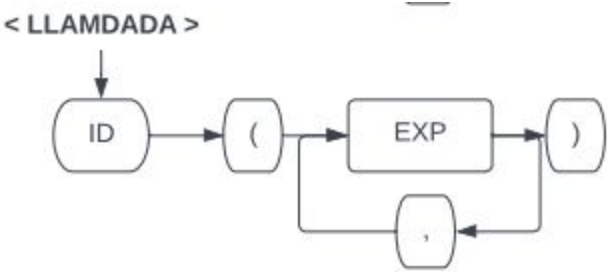
TIPO → int | float | char | bool



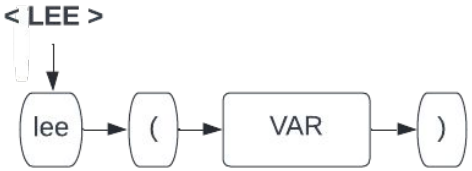
ESTATUTOS → ASIGNA | LLAMADA | LEE | ESCRIBE | CONDICION | CICLO-W | CICLO-F | FUNCIONES



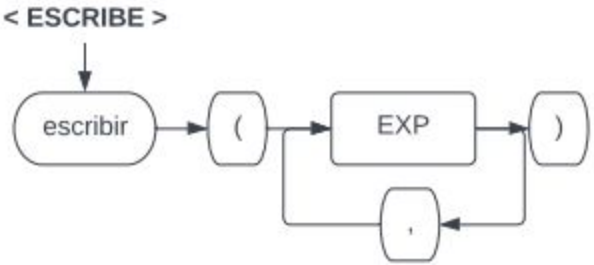
ASIGNA → VAR = EXP



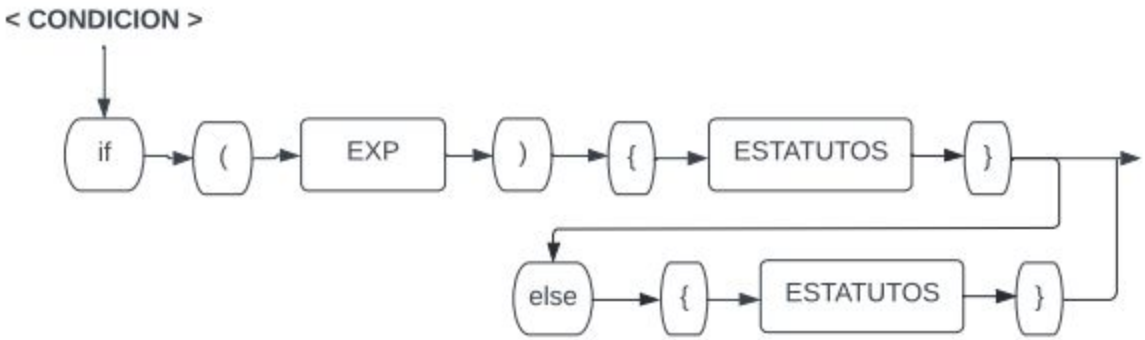
LLAMADA → ID (EXP LLAMADA2
LLAMADA2 → , EXP |)



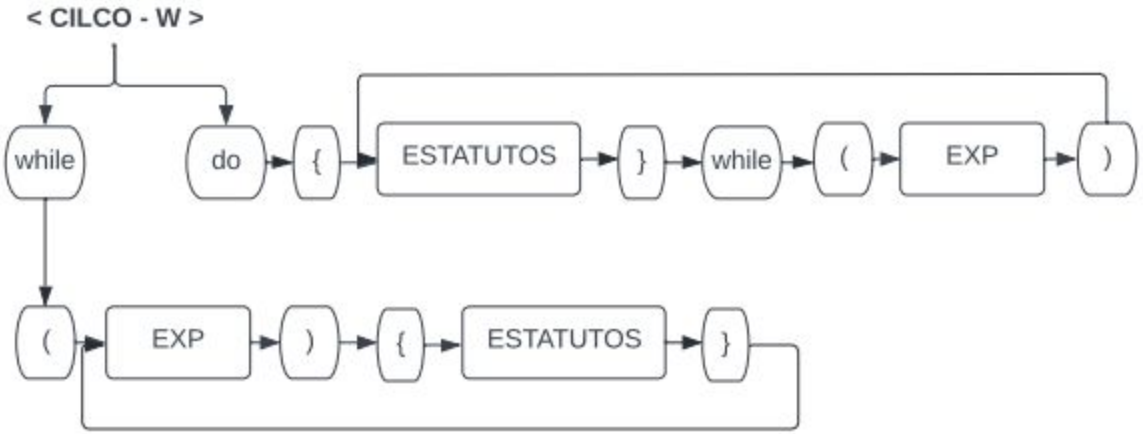
LEE → lee (VAR)



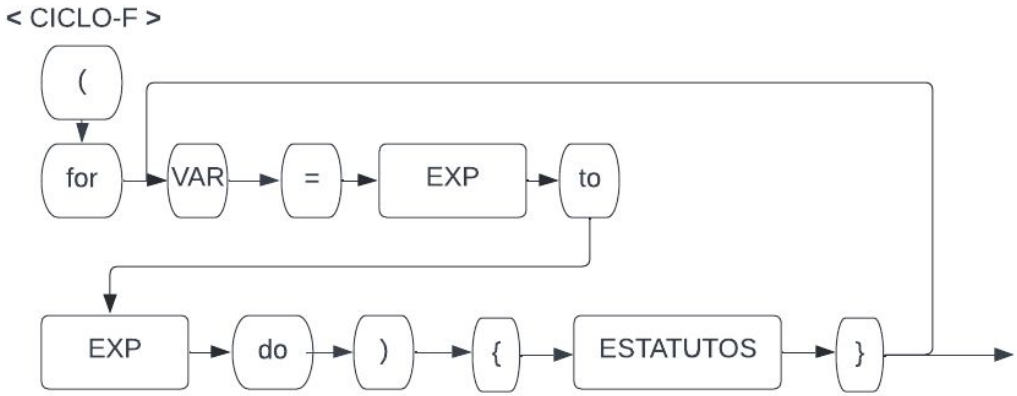
ESCRIBE → escribir (EXP ESCRIBE2
ESCRIBE2 → , EXP |)



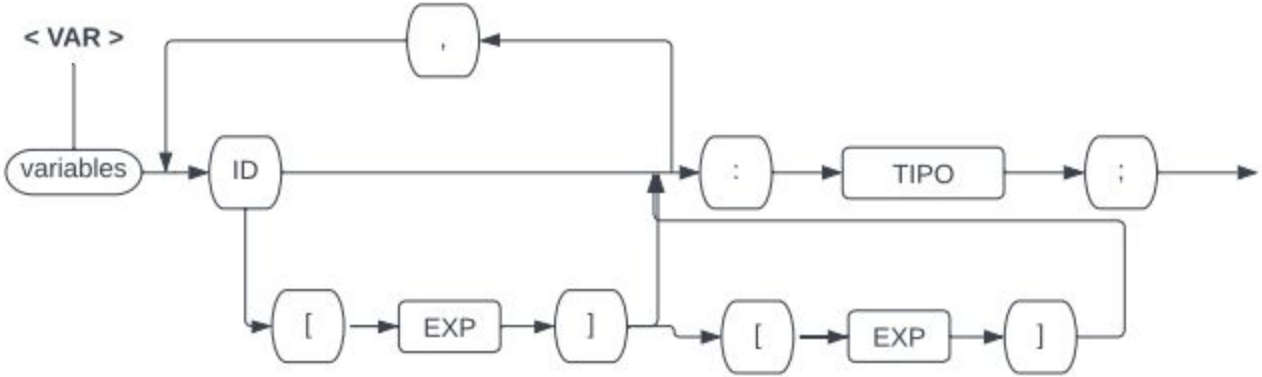
CONDICION → if (EXP) CONDICION2
CONDICION2 → { ESTATUTO } | else { ESTATUTO }



CICLO-W → while (CICLO-W2 | do { CICLO-W3
CICLO-W2 → EXP) { ESTATUTOS } CICLO-W2 | EXP) { ESTATUTOS }
CICLO-W3 → ESTATUTOS } while (EXP) CICLO-W3 | ESTATUTOS }
while (EXP)

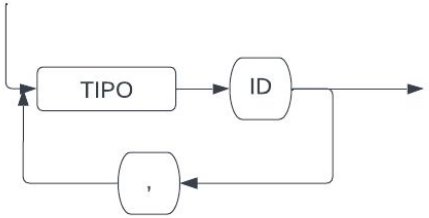


CICLO-F → (for CICLO-F2
CICLO-F2 → CICLO-F3 CICLO-F2 | CICLO-F3
CICLO-F3 → VAR = EXP to do { ESTATUTOS }

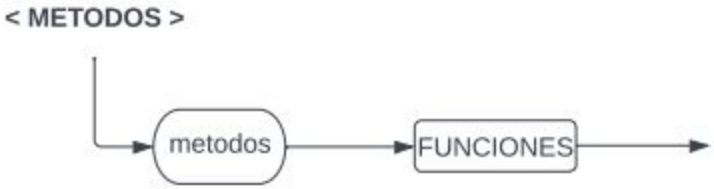


VAR → variables VAR2
VAR2 → ID VAR4 VAR3 | ID VAR3
VAR3 → , VAR2 | : TIPO ;
VAR4 → [EXP] | [EXP] [EXP]

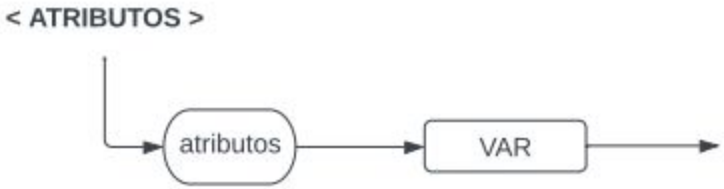
< PARAMETROS >



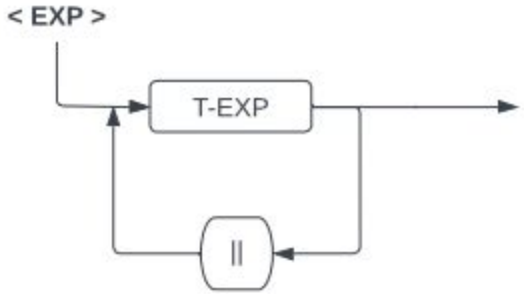
PARAMETROS → TIPO ID PARAMETROS2 | TIPO ID
PARAMETROS2 → , PARAMETROS



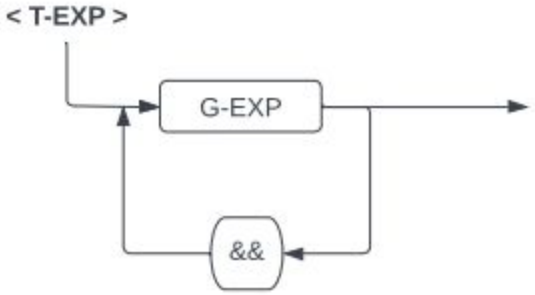
METODOS → metodos FUNCIONES



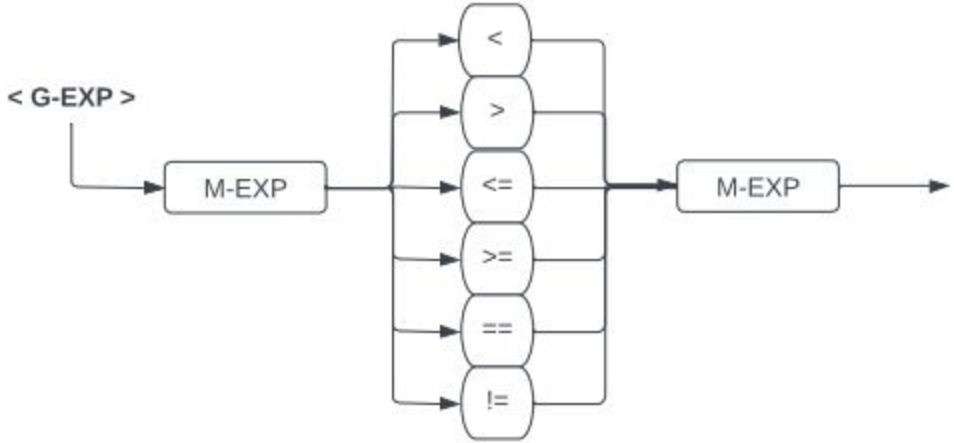
ATRIBUTOS → atributos VAR



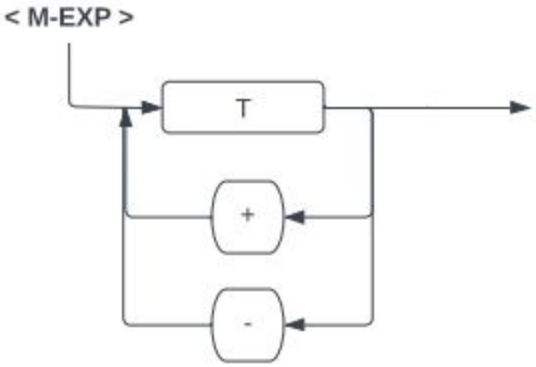
EXP → T-EXP | T-EXP || EXP



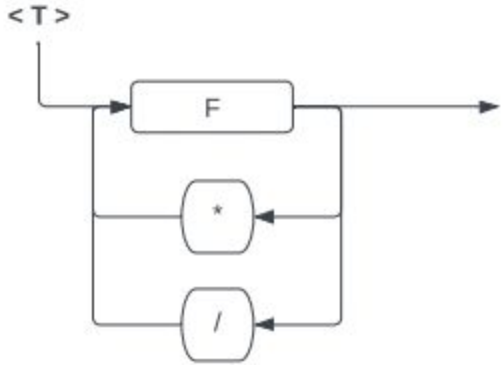
T-EXP → G-EXP | G-EXP && T-EXP



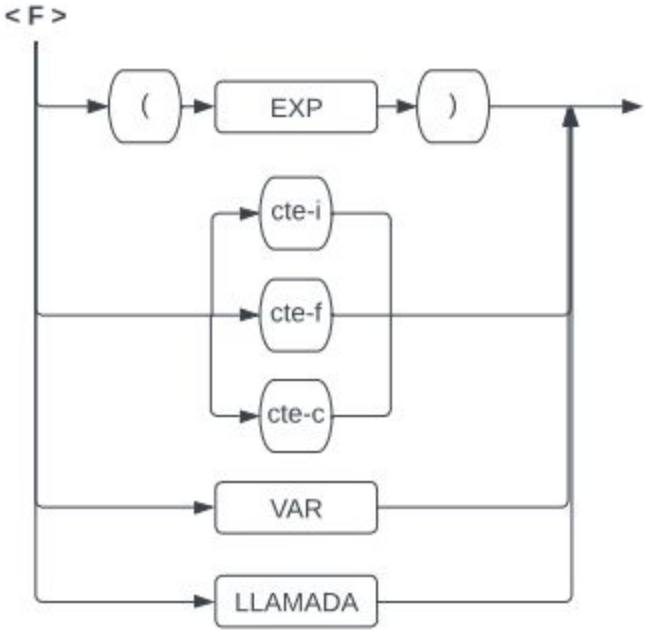
G-EXP → M-EXP G-EXP2 M-EXP
G-EXP2 → < | > | <= | >= | == | !=



M-EXP → T M-EXP2 | T
M-EXP2 → + M-EXP | - M-EXP



T → F T2 | F
T2 → * T | / T



F → (EXP) | F2 | VAR | LLAMADA
F2 → cte-i | cte-f | cte-c

1. **Programa:** Declarar el programa
2. **Principal:** Es la funcion Main() del programa
3. **Clases:** Declaración de clases
4. **Funciones:** Declaración de funciones
5. **Tipo:** Declaración de tipo
6. **Estatutos:** Llama a los diferentes posibles estatutos del programa
7. **Asigna:** Asigna valor a una variable
8. **Llamada:** Llamada de funciones
9. **Lee:** Lectura de variables
10. **Escribe:** Imprime el valor
11. **Condicion:** If - else
12. **Ciclo W:** While loop
13. **Ciclo F:** For loop
14. **Var:** Declaración de variables
15. **Parametros:** Declara los parámetros de una función
16. **Metodos:** Declara los métodos de la función
17. **Atributos:** Declara los atributos de la función
18. **Exp:** Condicional or
19. **Exp-T:** Condicional and
20. **G-Exp:** Resto de condicionales
21. **M-Exp:** Sumas y restas
22. **T:** Multiplicacion y division
23. **F:** Parentesis, declaración de variable o llamada