UNIVERSIDAD POLITÉCNICA DE CHIAPAS

INGENIERÍA EN SOFTWARE

8-A

2 CORTE

Proyecto: Analizador sintáctico descendente

MARÍA FERNANDA QUEZADA SUÁREZ
JULIO ADRIAN GALLEGOS BORRAZ
ADRIÁN MAURICIO HERNÁNDEZ PÉREZ
DANIEL EDUARDO BAUTISTA TRUJILLO

12/03/2025

1. Gramática Definida

Gramática

La gramática está diseñada para representar un lenguaje similar a C++, que incluye declaraciones, asignaciones, estructuras de control (if, while, for), y entradas/salidas (Cin, Printf). Los terminales representan palabras clave, identificadores, operadores y delimitadores, mientras que los no terminales definen las estructuras del lenguaje.

Programa → class main() { ListaInstrucciones }

ListaInstrucciones → Instruccion ListaInstruccionesOpcional

ListaInstruccionesOpcional → Instruccion ListaInstruccionesOpcional | ε

Instruccion → Declaracion | Asignacion | IfStatement | WhileStatement | ForStatement | EntradaStatement | SalidaStatement

Declaracion → TipoDato IDENTIFICADOR DeclaracionOpcional

DeclaracionOpcional → = ExpresionAritmetica ; | ;

Asignacion → IDENTIFICADOR = ExpresionAritmetica ;

IfStatement → if (Condicion) { ListaInstrucciones } ElseOpcional

ElseOpcional → else { ListaInstrucciones } | ε

WhileStatement → while (Condicion) { ListaInstrucciones }

ForStatement → for (Asignacion Condicion ; IDENTIFICADOR = ExpresionAritmetica) { ListaInstrucciones }

EntradaStatement → Cin (IDENTIFICADOR) ;

SalidaStatement → Printf (ArgPrintf) ;

TipoDato → int | float

ExpresionAritmetica → Termino ExpresionAritmeticaOpcional

ExpresionAritmeticaOpcional → + Termino ExpresionAritmeticaOpcional | - Termino ExpresionAritmeticaOpcional | ε

Termino → Factor TerminoOpcional

TerminoOpcional → * Factor TerminoOpcional | / Factor TerminoOpcional | ε

Factor → IDENTIFICADOR | NUMERO | NUMERO_FLOAT | (ExpresionAritmetica)

Condicion → ExpresionAritmetica OpRelacional ExpresionAritmetica

OpRelacional → < | > | <= | >= | == | ===

Cadena → " CadenaResto "

CadenaResto → IDENTIFICADOR | NUMERO CadenaResto | ε

2. Tabla de Análisis Predictivo

Una tabla de análisis predictivo se basa en los conjuntos FIRST y FOLLOW para predecir qué producción usar en cada paso. Aquí incluyo una parte de la tabla que debes completar basándote en los conjuntos FIRST y FOLLOW.

No terminal	Class	main ()	{	int	float	if	while	id	(	)	}	ε	$
Programa	P -> class main() { S }											ε	$
S				S -> int id	S -> float id	S -> if (E) S	S -> while (E) S				S -> ε		
E								E -> id op id					

Explicación:

- Para el no terminal P, cuando se encuentra el terminal class, se aplicará la producción P -> class main() { S }.
- Para el no terminal S, cuando se encuentra el terminal int, se aplicará la producción S -> int id, y de manera similar para float, if, y while.
- Para el no terminal S, si se encuentra un }, significa que puede derivar la producción vacía ε, lo que ocurre si el conjunto Follow de S contiene el terminal } o $.
- Para el no terminal E, cuando se encuentra el terminal id, se aplicará la producción E -> id op id.

3. Explicación del Funcionamiento del Analizador

El **Analizador Léxico** se encarga de convertir el código fuente en una secuencia de tokens. Usa expresiones regulares para reconocer palabras clave, identificadores, números, operadores, delimitadores y espacios, descartando estos últimos.

El **Analizador Sintáctico** aplica la gramática LL(1) para verificar si la secuencia de tokens sigue las reglas del lenguaje definido. Utiliza los conjuntos FIRST y FOLLOW para construir una tabla de análisis predictivo, que permite analizar la estructura del código de manera eficiente. El análisis es descendente recursivo, lo que significa que, dado un token de entrada, el analizador predice la regla gramatical a utilizar.

4. Casos de Prueba y Resultados

Puedes definir una serie de casos de prueba con fragmentos de código que el analizador debe procesar correctamente:

Caso 1: Declaración simple

int x;

Tokens generados:

- PALABRA_CLAVE -> int
- IDENTIFICADOR -> x
- DELIMITADOR -> ;

Resultado del análisis sintáctico:

- Reconocida la producción `Declaración → int IDENTIFICADOR ;`.

Caso 2: Expresión aritmética

x = 10 + 20;

Tokens generados:

- IDENTIFICADOR -> x
- ASIGNADOR -> =
- NUMERO -> 10
- OPERADOR -> +
- NUMERO -> 20
- DELIMITADOR -> ;

Resultado del análisis sintáctico:

- Reconocida la producción `Asignacion → IDENTIFICADOR = ExpresionAritmetica ;`.

Caso 3: Estructura condicional

```
if (x > 10) {

  Printf("Hola");

}
```

Tokens generados:

- PALABRA_CLAVE -> if
- DELIMITADOR -> (
- IDENTIFICADOR -> x
- OPERADOR_RELACIONAL -> >
- NUMERO -> 10
- DELIMITADOR ->)
- DELIMITADOR -> {

- PALABRA_CLAVE -> Printf
- DELIMITADOR -> (
- IDENTIFICADOR -> Hola
- DELIMITADOR ->)
- DELIMITADOR -> ;

Resultado del análisis sintáctico:

- Reconocida la producción `IfStatement → if ( Condicion ) { ListaInstrucciones } ElseOpcional`.

ANEXOS

Interfaz gráfica :