

BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

INTRODUCCION A LOS COMPILADORES

PROYECTO FINAL

PROFESORA:

Dra. Hilda Castillo Zacatelco

PRESENTA:

STEWART AXEL GOMEZ LOZADA

NOVIEMBRE 2018

1.-INTRODUCCION

A lo largo de este documento veremos el desarrollo del proyecto final de la materia de introducción a los compiladores. Tendremos una vista a los diferentes tokens que acepta el lenguaje, también veremos cual fue la gramática utilizada para realizar el análisis sintáctico ademas de las estructuras utilizadas para la interpretación.

Para ejemplificar mejor algunos aspectos se utilizaran diagramas y partes de código.

2.-ANALISIS

Ahora veremos los elemento utilizados para realizar cada una de las partes del análisis de nuestro lenguaje.

2.1-ANALISIS LEXICO

Para el análisis léxico se utilizo un autómata finito no determinista. De acuerdo a las especificaciones del lenguaje se concluyo que estos serian los tokens que el lenguaje aceptaría:

1. Suma
2. Resta
3. Por
4. Entre
5. Menor o igual
6. Menor
7. Mayor o igual
8. Mayor
9. Igual
10. Asignación
11. Diferente
12. Carácter
13. Coma
14. Llaves que abren
15. Llaves que cierran
16. Paréntesis que abre
17. Paréntesis que cierra
18. Punto y coma
19. Corchete que abre
20. Corchete que cierra
21. Entero
22. Id
23. Punto

Nuestras palabras reservadas serán las siguientes:

Programa Si

Constantes Entonces

Arreglos Sino

Inicio Escribe

Fin Lee

Length Mod

Para

Hasta

Paso

hacer

Al diseñar el autómata para que se aceptaran los tokens correspondientes resulto el diagrama 2.1.1:

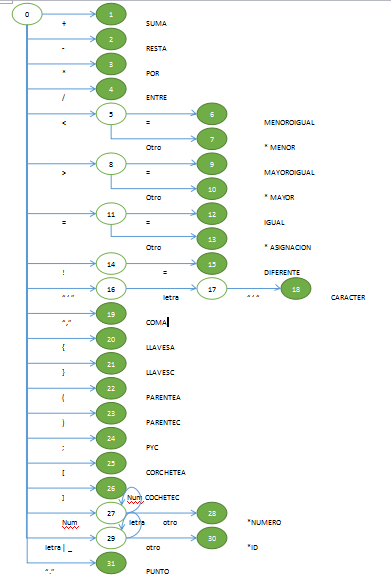


DIAGRAMA 2.1.1 Muestra la disposición el autómata utilizado para el analizador léxico

2.2 ANALISIS SINTACTICO

Para el análisis sintáctico se hizo uso de gramáticas. Para este lenguaje se utilizo una gramática LL1. A continuación veremos el diseño de la gramática LL1.

De acuerdo a las especificaciones del lenguaje se diseño la siguiente gramática:

1. > programa id DC DA inicio INS fin

DC-> constantes id = VAL CONS

DC->δ

CONS->id = VAL CONS

CONS->δ

DA-> arreglos id={ARVAL} ARR

DA->δ

ARR-> id = { ARVAL} ARR

ARR->δ

ARVAL-> VAL ARVAL2

ARVAL2-> , VAL ARVAL2

ARVAL2->δ

VAL-> entero

VAL->carácter

INS-> EXP INS

INS-> SI INS

INS-> PARA INS

INS-> LEE INS

INS->ESCRIBE INS

INS->δ

EXP-> VARIABLE = EXPRESION;

VARIABLE-> id INDI

INDI->[ EXPRESION ]

INDI->δ

EXPRESION->VARCON EXPRESION2

EXPRESION2-> OP VARCON

EXPRESION2->δ

VARCON-> id VC

VARCON-> VAL

VC-> [ EXPRESION ]

VC-> . length

VC->δ

1. > +

P-> -

OP->P

OP->\*

OP->/

OP-> mod

SI-> si ( CONDICION ) entonces SINO

SINO-> fin

SINO-> sino INS fin

CONDICION-> VARCON CON VARCON

CON-> ==

CON-> !=

CON-> >

CON-> >=

CON-> <

CON-> <=

PARA-> para VARIABLE = INTEGER hasta INTEGER paso P entero hacer INS fin

INTEGER-> id VC

INTEGER-> entero

LEE-> lee ( VARIABLE ) ;

ESCRIBE-> escribe ( VARCON ) ;

Una vez realizada la gramática, calculamos el conjunto predictivo y nos aseguramos que la gramática sea LL1. A continuación calcularemos el conjunto predictivo de las producciones

S-> programa id DC DA inicio INS fin

{programa}

DC->constantes id = VAL CONS

{constantes}

DC->δ

{arreglos, inicio}

CONS->id = VAL CONS

{id}

CONS->δ

{arreglos, inicio} //parte corregida, se agregó inicio

DA-> arreglos id={ARVAL} ARR

{arreglos}

DA->δ

{inicio}

ARR-> id = { ARVAL} ARR

{id}

ARR->δ

{inicio}

ARVAL-> VAL ARVAL2

{entero,caracter}

ARVAL2-> , VAL ARVAL2

{“ , ”}

ARVAL2->δ

{ “ } ”}

VAL-> entero

{entero}

VAL->carácter

{caracter}

INS-> EXP INS

{id}

INS-> SI INS

{si}

INS-> PARA INS

{para}

INS-> LEE INS

{lee}

INS->ESCRIBE INS

{escribe}

INS->δ

{fin} // el no terminal “sino” no pertenece al conjunto predictivo

EXP-> VARIABLE = EXPRESION;

{id}

VARIABLE-> id INDI

{id}

INDI->[ EXPRESION ]

{“ [ “}

INDI->δ

{=}

EXPRESION->VARCON EXPRESION2

{id, entero, caracter}

EXPRESION2-> OP VARCON

{+,-,\*,/,mod}

EXPRESION2->δ

{ “;”,“ ] ”}

VARCON-> id VC

{id}

VARCON-> VAL

{“ [ ”}

{entero, caracter}

VC-> [ EXPRESION ]

{ “ [ “}

VC-> . length

{ “ . “}

VC->δ

{==,!=,>=,>,<=,<,” ] ”, +,-,\*,/,mod,” ; ”, hasta, paso,”)”} // Se agregó el paréntesis que cierra

P-> +

{+}

P-> -

{-}

OP->P

{+,-}

OP->\*

{\*}

OP->/

{/}

OP-> mod

{mod}

SI-> si ( CONDICION ) entonces SINO

{si}

SINO-> fin

{fin}

SINO-> sino INS fin

{sino}

CONDICION-> VARCON CON VARCON

{id,entero,caracter}

CON-> ==

{==}

CON-> !=

{!=}

CON-> >

{>}

CON-> >=

{>=}

CON-> <

{<}

CON-> <=

{<=}

PARA-> para VARIABLE = INTEGER hasta INTEGER paso P entero hacer INS fin

{para}

INTEGER-> id VC

{id}

INTEGER-> entero

{entero}

LEE-> lee ( VARIABLE ) ;

{lee}

ESCRIBE-> escribe ( VARCON ) ;

{escribe}

Como nos podemos dar cuenta las diferentes producciones de un símbolo no terminal tienen conjuntos predictivos diferentes, por lo tanto, podemos concluir que esta gramática es LL1. Como nuestra gramática diseñada es LL1 podemos implementarla de manera sencilla. Aquí tenemos el ejemplo del símbolo inicial:

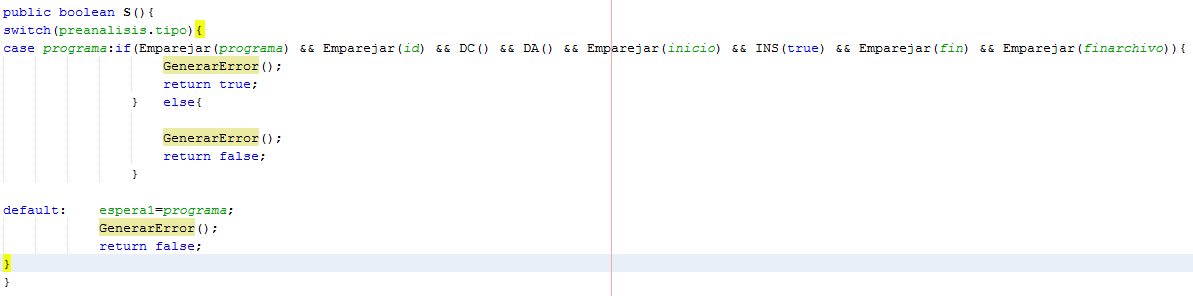


Figura 2.2.1 muestra el código que implementa del símbolo inicial

3.-DESARROLLO

3.1 ANALIS LEXICO

La parte del análisis léxico fue echa con la herramienta que aviamos creado anteriormente. Se creo un archivo llamado “Especificacion\_Lexico.txt” dentro de la carpeta del proyecto. Cada vez que se inicia el programa se abre el archivo y se carga la especificación. La especificación es la misma que ya habíamos mostrado anteriormente.

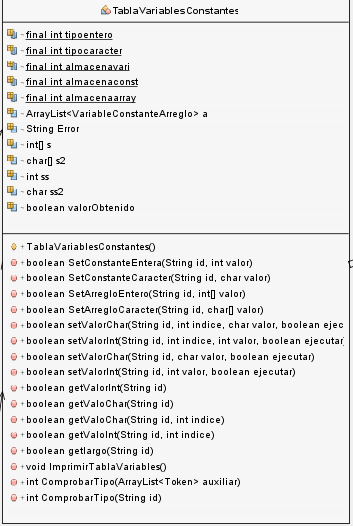
3.2 ANALISIS SINTACTICO

Para el análisis sintáctico utilizamos el método de mostrado en la clase.

* 1. Cada símbolo no terminal es una función.
  2. Existe una función de emparejamiento.
  3. Se utilizan el conjunto predictivo para saber qué producción realizara un símbolo no terminal.

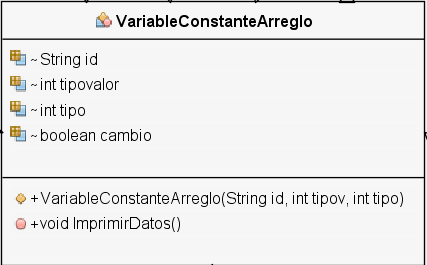
3.3 ANALISIS SEMANTICO E INTERPRETACION

Para la creación de variables, constantes y arreglos se utiliza una clase llamada “TablaVariablesConstantes”, y está organizada de la siguiente forma



Como se puede observar esta clase almacena un arraylist de VariableConstanteArreglo la cual es nuestra clase padre de los datos a almacenar. La clase VariableConstanteArreglo contiene dos atributos:

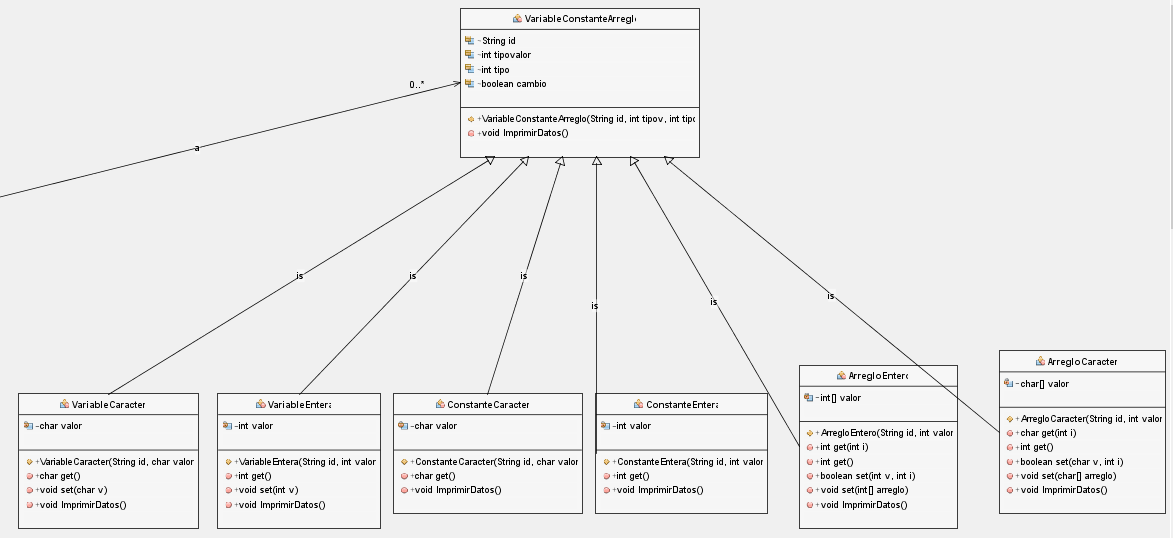
* tipovalor (este atributo nos indica el tipo de dato que se almacena)
* tipo (este atributo nos indica que tipo es; variable, constante o arreglo).



De esta clase de desprenden 6 subclases;

* VariableCaracter
* VariableEntera
* ConstanteCaracter
* ConstanteEntera
* ArregloCaracter
* ArregloEntero.

Cada uno permite hacer operaciones dependiendo del tipo de dato y el tipo de almacenamiento.



También existe una clase llamada Datos, esta clase se utiliza para mandar información entre funciones.



4.-CONCLUCIONES

La implementación del lenguaje fue exitosa, utilizando autómatas, y gramáticas para realizar el análisis léxico y análisis sintáctico respectivamente. A pesar que la parte gráfica necesita mejoras, por ejemplo: una mejor organización y presentación, remarcar que elemento especifico del arreglo fue modificado, y mostrar de forma mas comprensible la linea que se esta ejecutando. Se puede decir que esta primera versión cumple de forma satisfactoria los requisitos impuestos.

Como trabajo a largo plazo se piensa realizar las mejoras en los aspectos antes remarcados. También se pensara en poner a prueba el lenguaje con alumnos, para obtener una retroalimentacion de posibles mejoras que se puedan implementar

5.-REFERENCIAS

Aho, A., Lam, M., Sethi, R., & Ullman, J. (2015). *Compilers*. India: Pearson India Education Services.