UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INGENIERÍA EN COMPUTACIÓN

COMPILADORES

ANALIZADOR SINTÁCTICO

GRUPO:02

FECHA DE ENTREGA LÍMITE: 07*/NOVIEMBRE/2017*

EQUIPO DE TRABAJO:

- CHÁVEZ DELGADO JORGE LUIS
- SÁNCHEZ NERI DAVID YAXKIN

SEMESTRE 2018-1

INDICE

1.- DESCRIPCIÓN DEL PROBLEMA

Se construirá un analizador sintáctico descendente, derivado del analizador léxico elaborado en Flex y ANSI C definido por la gramática planteada en clase (*Ir a diseño e implementación*) y la tabla de componentes léxicos que se muestra a continuación:

Clase	Descripción	Átomo(s)
0	Identificadores (sólo letras minúsculas)	a
1	Palabras reservadas	
2	Operador de asignación =	=
3	Operadores relacionales	
4	Operadores aritméticos	
5	Símbolos especiales , ; [ ] ( ) @	, ; [ ] ( ) @
6	Constantes numéricas enteras (base 10)	c
7	Constantes numéricas reales (siempre con .)	n
8	Constante cadena (entre comillas)	s

2.- PROPUESTA DE SOLUCIÓN

En el analizador léxico, se almacenaban ya los tokens de las clases, por lo que se añadieron también las instrucciones para que agregara los caracteres correspondientes a los átomos de los componentes léxicos reconocidos. Para lograr tal objetivo, los arreglos utilizados para las tablas estáticas se transformaron en matrices, generando una especie de “diccionario” con la palabra reservada y su átomo.

3.- ÁNALISIS

Analizando las diferentes formas de realizar el analizador, hemos decidido utilizar la recursión que nos proporcionan las funciones del lenguaje C, debido a que es mucho más sencillo analizarlo de esta manera, pero poco más complicado escribir el código sin equivocarse. Por otro lado, las estructuras nos ayudarán a mantener presentes cada uno de nuestros átomos.

El primer paso es calcular los conjuntos de selección para poder realizar cada una de las funciones en lenguaje C, para posteriormente del archivo que ya recibía el analizador léxico obtener la cadena de átomos que genera y analizarla sintácticamente mediante la función ***parser()*** la cual desencadena en recursión de cada una de las funciones de las reglas de producción (*Ir a diseño e implementación*).

4.-CONJUNTOS DE SELECCIÓN

Observando que los conjuntos de selección de las producciones cuyo lado izquierdo es el mismo son disjuntos, concluimos que la gramática es una gramática LL (1):

#	Regla de producción	Conjunto de selección
1.-	P→<LF>	**ıtra**
2.-	<LF>→ε	**ı**
3.-	<LF>→<FUN><LF>	**tra**
4.-	<FUN>→Va(<LA>)[<LD><BP>]	**tr**
5.-	<FUN>→a(<LA>)[<LD><BP>]	**a**
6.-	<LA>→ε	**)**
7.-	<LA>→Va<LAP>	**tr**
8.-	<LAP>→ε	**)**
9.-	<LAP>→,Va<LAP>	**,**
10.-	<LD>→ε	**]@awlhmi[**
11.-	<LD>→D<LD>	**tr**
12.-	D→VaCL	**tr**
13.-	V→t	**t**
14.-	V→r	**r**
15.-	C→ε	**,;**
16.-	C→=N	**=**
17.-	N→n	**n**
18.-	N→c	**c**
19.-	L→,aCL	**,**
20.-	L→;	**;**
21.-	<BP>→ε	**]**
22.-	<BP>→<PR><BP>	**@awlhmi[**
23.-	<PR>→S	**@awlhmi**
24.-	<PR>→<PC>	**[**
25.-	S→A	**a**
26.-	S→W	**w**
27.-	S→R	**l**
28.-	S→H	**h**
29.-	S→M	**m**

30.-	S→I	**i**
31.-	S→@a(<LP>);	**@**
32.-	<PC>→[<BP>]	**[**
33.-	A→a=E;	**a**
34.-	W→wW'	**w**
34.1.-	W'→aW''	**a**
34.2.-	W'→cW''	**c**
34.3.-	W'→nW''	**n**
34.4.-	W'→sW''	**s**
34.5.-	W''→,W'	**,**
34.6.-	W''→;	**;**
35.-	R→laR'	**L**
35.1.-	R'→,aR'	**,**
35.2.-	R'→;	**;**
36.-	H→h[<BP>]m(<REL>)	**h**
37.-	M→m(<REL>)<PR>	**m**
38.-	I→i(<REL>)<PR>e<PR>	**i**
39.-	<REL>→E<OR>E	**(anc@**
40.-	<OR>→>	**>**
41.-	<OR>→g	**g**
42.-	<OR>→<	**<**
43.-	<OR>→p	**p**
44.-	<OR>→q	**q**
45.-	<OR>→!	**!**
46.-	<LP>→E<LPA>	**(anc@**
47.-	<LP>→ε	**)**
48.-	<LPA>→ε	**)**
49.-	<LPA>→,E<LPA>	**,**
50.-	E→TE'	**(anc@**
51.-	E'→+TE'	**+**
52.-	E'→-TE'	**-**
53.-	E'→ε	**,);>g<pq!**
54.-	T→FT'	**(anc@**
55.-	T'→*FT'	*****
56.-	T'→/FT'	**/**
57.-	T'→ε	**,-+);>g<pq!**
58.-	F→(E)	**(**
59.-	F→a	**a**
60.-	F→n	**n**

61.-	F→c	**c**
62.-	F→@a(<LP>)	**@**

5.- ¿CÓMO CORRER EL PROGRAMA?

Para este proceso necesitamos una terminal linux, y tener instalado el compilador gcc y flex, en caso de no tenerlos instalados, en sistemas operativos basados en Debian se instalaría de la siguiente manera:

Para gcc:

$ sudo apt-get install gcc

Para flex:

$ sudo apt-get install flex

Teniendo ya instalado lo anterior procedemos a ejecutar nuestro código escrito en flex y lenguaje C de la siguiente manera:

$ flex analizadorS.l

La salida al ejecutar el comando anterior nos devolverá un archivo llamado ***lex.yy.c*** el cual podrá ser compilado con ayuda de gcc de la siguiente forma:

$ gcc lex.yy.c -lfl

El resultado de la compilación nos generará un archivo ejecutable el cual podremos correr y visualizar el análisis léxico de nuestro futuro compilador, para esto se utilizó un archivo de prueba llamado “entrada” de la siguiente manera:

$ ./a.out entrada

6.-CONCLUSIONES

Chávez Delgado Jorge Luis:

Se puede concluir que es necesario e indispensable saber calcular los conjuntos de selección para poder realizar un correcto análisis sintáctico, ya que de no ser así el proceso de construcción puede ser un poco más tardado debido a que el proceso se va tornando como prueba y error. Además, mientras avanzamos en el programa, nos dimos cuenta que, si bien es sencillo realizar la recursión, las líneas de código son mayores y esto aumenta la posibilidad de generar errores y no encontrarlos fácilmente.

Sánchez Neri David Yaxkin:

Durante la creación del programa se terminó de comprender la utilidad del cálculo de los conjuntos de selección para las reglas de derivación de nuestra gramática, ya que facilitaron en gran medida la creación de las funciones para el parser y la depuración, que se complica por la gran cantidad de estructuras de selección (***if***) que se utilizaron, ya que tuvimos algunos problemas con algunas reglas de derivación, con lo cual bastaba con checar si nuestros conjuntos de selección estaban determinados de manera correcta.