UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INGENIERÍA EN COMPUTACIÓN

COMPILADORES

ANALIZADOR SINTÁCTICO

GRUPO:02

FECHA DE ENTREGA LÍMITE: 08*/DICIEMBRE/2017*

EQUIPO DE TRABAJO:

- CHÁVEZ DELGADO JORGE LUIS
- SÁNCHEZ NERI DAVID YAXKIN

SEMESTRE 2018-1

INDICE

1.- DESCRIPCIÓN DEL PROBLEMA

Se construirá un analizador léxico, sintáctico descendente, derivado del analizador léxico elaborado en Flex; además se revisará con ayuda del traductor, programas escritos en el lenguaje Cfleliz.Todo esto escrito en lenguaje 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

Para los procesos semánticos:

- Se manejará los atributos, apoyándose en la cadena generada por el análisis léxico, y se utilizará una función para devolver el tipo que se encuentra almacenado en la tabla de palabras reservadas.
- Se utilizará una función para obtener el tipo de los tokens en caso de que se repitan.
- Si el identificador no fue declarado, se mostrará un mensaje de error.

Para el proceso de traducción:

- Se añadirán los símbolos de acción a la gramática para poder realizar la traducción, y se utilizarán dos funciones, ambas se apoyarán en la tabla de tokens para realizar la búsqueda en las tablas de símbolos y de cadenas.
- La traducción se realiza a la par que el análisis sintáctico, al avanzar un carácter de la cadena de átomos se avanzará también un token de la tabla de tokens.

3.- ÁNALISIS

- Se analizó la gramática utilizada para el analizador léxico-sintáctico, y se decidió colocar los atributos en las reglas de producción correspondientes a la declaración de funciones, variables y lista de argumentos de las funciones.}
- Se colocaron los símbolos de acción correspondientes a cada uno de los terminales de la gramática de los cuales es necesario escribir el valor leído del programa fuente al programa de salida.
- La excepción fueron los operadores aritméticos, ya que estos tenían que ser traducidos, la palabra (MAS MENOS MULTIPLICA DIVIDE) en los símbolos (+ - * / respectivamente).
- Se añadieron saltos de línea a los símbolos de acción como **; [ ]**
- Todo lo anterior se facilitó debido al uso de listas ligadas para guardar la tabla de tokens.

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), además también se incluyen los símbolos de acción, así como los atributos:

#	Regla de producción con símbolo de acción y atributos	Conjunto de selección
1.-	P→<LF>	**ŀtra**
2.-	<LF>→ε	**ŀ**
3.-	<LF>→<FUN><LF>	**tra**
4.-	<FUN>→$V_t a_p${a}{AT}$_{t1,p1}$({(}<LA>){)}[{[}<LD><BP>]{]}	**tr**
5.-	<FUN>→a_p{a}{AT}$_{-1,p1}$({(}<LA>){)}[{[}<LD><BP>]{]}	**a**
6.-	<LA>→ε	**)**
7.-	<LA>→ $V_t a_p${a}{AT}$_{t1,p1}$<LAP>	**tr**
8.-	<LAP>→ε	**)**
9.-	<LAP>→,{,} $V_t a_p${a}{AT}$_{t1,p1}$<LAP>	**,**
10.-	<LD>→ε	**]@awlhmi[**
11.-	<LD>→D<LD>	**tr**
12.-	D→ $V_t a_p${a}{AT}$_{t1,p1}CL_t$	**tr**
13.-	V_v→t_t{t}	**t**
14.-	V_v→r_t{r}	**r**
15.-	C→ε	**,;**
16.-	C→={=}N	**=**
17.-	N→n{n}	**n**
18.-	N→c{c}	**c**
19.-	L_t→,{,}a_p{a}{AT}$_{t1,p1}CL_t$	**,**
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	**l**
31.-	S→@{@}a{a}({(}<LP>){)};{;}	**@**
32.-	<PC>→[{[}<BP>]{]}	**[**
33.-	A→a{a}={=}E;{;}	**A**

34.-	W→w{w}W’	**W**
34.1.-	W’→a{a}W’’	**A**
34.2.-	W’→c{c}W’’	**C**
34.3.-	W’→n{n}W’’	**N**
34.4.-	W’→s{s}W’’	**S**
34.5.-	W’’→,{,}W’	**,**
34.6.-	W’’→;{;}	**;**
35.-	R→laR’	**L**
35.1.-	R’→,{,}aR’	**,**
35.2.-	R’→;{;}	**;**
36.-	H→h{h}[{[}<BP>]{]}m{m}({(}<REL>){)};{;}	**H**
37.-	M→m{m}({(}<REL>){)}<PR>	**M**
38.-	I→i{i}({(}<REL>){)}<PR>e{e}<PR>	**I**
39.-	<REL>→E<OR>E	**(anc@**
40.-	<OR>→>{>}	**>**
41.-	<OR>→g{g}	**g**
42.-	<OR>→<{<}	**<**
43.-	<OR>→p{p}	**p**
44.-	<OR>→q{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}	**a**
60.-	F→n{n}	**n**
61.-	F→c{c}	**c**
62.-	F→@{@}a{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 analizadorSyT.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, si no se tienen claros los conceptos de símbolos de acción y atributos, no se puede realizar correctamente la traducción ya que no se sabría en dónde colocarlos y además encadenaría errores al momento de programar.
Por otro lado, ya teniendo los símbolos de acción y atributos en la gramática es más sencillo manejar las funciones con el analizador sintáctico recursivo para poder aplicar la traducción y actualización del tipo.

Sánchez Neri David Yaxkin:

Al realizar el programa se encontraron algunos retos, como: en dónde y cómo colocar los símbolos de acción y atributos, y cómo relacionarlos con la estructura actual de nuestro programa, por lo que se tuvo que repasar los conceptos vistos en clase para poder realizar esta tarea. Después de haber revisado la nueva gramática se pudo realizar el programa de una manera fluida, quedándonos solo con algunos errores lógicos que también nos ayudaron a encontrar algunos errores o atributos faltantes en nuestra gramática.