



Universidad Nacional
Autónoma de México
Facultad de Ingeniería



Compiladores
Clave 434

Analizador sintáctico

Alumno:
Ángel Alvarado Campos

Grupo 3
Semestre 2023-1

Profesora: M.C. Laura Sandoval Montaña

Descripción del problema

El proyecto de la asignatura de Compiladores consiste en desarrollar un compilador para un lenguaje definido por el grupo, para lo cual es necesario dividir el problema en varias etapas. En una entrega anterior, se abordó la etapa de *análisis léxico*, en la cual se definieron en Lex/Flex los componentes léxicos del lenguaje en cuestión. En esta segunda entrega, se propone desarrollar la etapa de *análisis sintáctico* del lenguaje, la cual consiste en definir la gramática del lenguaje en cuestión.

En sí, con las funciones especiales de Lex/Flex se puede garantizar el reconocimiento de cadenas a partir de ciertos patrones (expresiones regulares) definidos, lo cual corresponde a la definición de componentes léxicos, sin embargo, para implementar la sintaxis del lenguaje (es decir, la gramática), Lex/Flex no promueve alguna función especial para esto. No obstante, Lex/Flex trabaja con el lenguaje de programación C, así que por medio de funciones en C sí es posible definir a la gramática del lenguaje a partir de los componentes léxicos definidos. A pesar de lo anterior, con el reconocimiento de clases de Lex, con una pequeña modificación en el programa anterior se pudo generar una *cadena de átomos* que es entrada para la etapa de análisis sintáctico.

Existen dos enfoques para definir a la gramática del lenguaje de manera *algorítmica*: analizador descendente predictivo y analizador descendente recursivo. En este caso, se solicitó tomar el enfoque de *analizador descendente recursivo* para implementar a la gramática del lenguaje. Con el analizador descendente recursivo, para cada elemento no terminal de la gramática se define una única función, cuyo comportamiento depende de la naturaleza de las reglas de producción en las que el elemento no terminal está en el lado izquierdo. Cuando se refiere a *la naturaleza* de la producción, se hace referencia a la estructura del lado derecho de la producción y, por lo tanto, del conjunto de selección correspondiente.

Para implementar al analizador descendente recursivo en lenguaje C fue necesario, en primer lugar, hacer el cálculo de los conjuntos de selección, lo cual fue un proceso largo que se describe con detalle más adelante. Posterior al cálculo de los conjuntos de selección, se realizó la definición de las funciones auxiliares para cada no terminal. Cada función de elemento no terminal debe definir los casos suficientes, correspondientes a cada producción en la que el no terminal es inicial. Estos casos deben definirse como condicionales respecto a la pertenencia de un carácter en el conjunto de selección de cada producción; esto se hace con estructuras *if-else*. Por otro lado, dentro de cada condición, que a su vez refiere a una única producción, se establecen ciertas operaciones en función de lo que se encuentra en el lado derecho de esa producción. Las operaciones referidas se muestran en la siguiente tabla.

Caso	Tipo de producción	Condición	Operación
1	$i: A \rightarrow b$	$if\ c == b$	$c = get_char()$ $return$

2	$i: A \rightarrow b\alpha$	$if\ c == b$	$c = get_char()$ $procesa(\alpha)$ $return$
3	$i: A \rightarrow \varepsilon$	$if\ c \in CS(i)$	$Return$
4	$i: A \rightarrow B\alpha$	$if\ c \in CS(i)$	$procesa(B\alpha)$ $return$

La operación $get_char()$ de la tabla anterior obtiene al siguiente carácter de la cadena de átomos.

De la tabla, se supone que b es un elemento terminal, B es un elemento no terminal, y α es cualquier subcadena, formada por uno o más elementos terminales o no terminales.

La operación $procesa(\alpha)$, por cada elemento no terminal que se encuentre en α , realiza una llamada a la función auxiliar correspondiente; por otro lado, por cada elemento terminal k que se encuentre en α , se evalúa la condicional $if\ (c == k)$. Si la condicional tiene un valor de verdad de verdadero, entonces se realiza la operación $c = get_char()$, en caso contrario, se realiza una llamada a la función $rechaza()$, con lo cual se indica que la cadena de entrada es sintácticamente incorrecta.

Modificaciones realizadas al analizador léxico

Respecto a la entrega anterior, en la etapa de analizador léxico cambió el tratamiento de errores. En la primera entrega los errores eran tratados de manera particular para cadenas de caracteres, números, símbolos, etc; lo cual traía problemas al momento de colocar a dos componentes léxicos de manera contigua, pues, por ejemplo, la cadena \$var4, era interpretada como un *error de identificador* porque los identificadores no pueden contener números, cuando en realidad la cadena se debía interpretar como \$var, un identificador, seguido de 4, un entero. Otro ejemplo similar se presentaba con \$4symbol, pues igualmente era tomada como error, cuando en realidad debía identificar a \$ como un identificador erróneo, 4 como un entero, y symbol como palabra reservada.

Para corregir esto, se hizo lo sugerido en la rúbrica calificada del programa anterior: utilizar el operador . (punto final) para controlar los errores. Entonces, se eliminó a todas las expresiones regulares correspondientes a casos particulares de diferentes clases y se definió una única expresión regular de error con el operador, con la menor *prioridad* posible.

Cálculo de los conjuntos de selección

En primer lugar, por inspección, se identificaron los elementos terminales y no terminales de la gramática.

No terminales	Terminales
<Program>	a
<otraFunc>	b
<Func>	f
<Param>	t
<otroParam>	g
<Cuerpo>	w
<Decl>	m
D	#
<Tipo>	o
K	x
Q	j
B	h
C	p
A	c
A'	q
E	y
E'	z
T	<
T'	>
F	l
R	u
R'	e
V	d
V'	i
V''	n
V'''	r
P	s
<ListaP>	[
W]
I	(
I')
J	{
Y	}
X	,
Z	:
H	;
C'	+
O'	-
U	*

<Devuelve>	/
<valor>	%
<Llama>	\
<arg>	^
<otroArg>	=

Se identificaron los elementos no terminales *anulables*:

- <otraFunc>
- <Param>
- <otroParam>
- <Cuerpo>: Es terminal porque su lado derecho está formado por <Decl> y <listaP>, únicamente, los cuales también son no terminales anulables
- <Decl>
- Q
- C
- E'
- T'
- <listaP>
- I'
- C'
- O'
- U
- <valor>
- <arg>
- <otroArg>

Se calculó el conjunto *first* de cada producción.

$$First(1) = first(< Func >) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$$First(2) = first(< Func >) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$$First(3) = \{\}$$

$$First(4) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$$First(5) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$$First(6) = \{\}$$

$$First(7) = \{,\}$$

$$First(8) = \{\}$$

$$First(9) = first(< Decl >) \cup first(< ListaP >) = first(D) \cup first(< ListaP >) \\ = first(< Tipo >) \cup first(< ListaP >) = \{b\ g\ \# \ y\ x\ c\ i\ f\ h\ w\ j\ [z\}$$

$$First(10) = \{\}$$

$$First(11) = first(D) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$$First(12) = first(< Tipo >) = \{b\ g\ \# \ y\ x\}$$

$First(13) = \{b\}$
 $First(14) = \{g\}$
 $First(15) = \{\#\}$
 $First(16) = \{y\}$
 $First(17) = \{x\}$
 $First(18) = \{i\}$
 $First(19) = \{\}$
 $First(20) = \{=\}$
 $First(21) = \{,\}$
 $First(22) = \{n\}$
 $First(23) = \{r\}$
 $First(24) = \{s\}$
 $First(25) = \{\}$
 $First(26) = \{,\}$
 $First(27) = \{i\}$
 $First(28) = \{s\}$
 $First(29) = first(E) = first(T) = first(F) = \{(i n r []\}$
 $First(30) = first(T) = first(F) = \{(i n r []\}$
 $First(31) = \{+\}$
 $First(32) = \{-\}$
 $First(33) = \{\}$
 $First(34) = first(F) = \{(i n r []\}$
 $First(35) = \{*\}$
 $First(36) = \{/ \}$
 $First(37) = \{\backslash\}$
 $First(38) = \{\%\}$
 $First(39) = \{^\wedge\}$
 $First(40) = \{\}$
 $First(41) = \{(\}$
 $First(42) = \{i\}$
 $First(43) = \{n\}$
 $First(44) = \{r\}$
 $First(45) = first(< llama >) = \{[]\}$
 $First(46) = \{i\}$
 $First(47) = \{n\}$
 $First(48) = \{r\}$
 $First(49) = \{s\}$
 $First(50) = \{>\}$
 $First(51) = \{<\}$
 $First(52) = \{l\}$
 $First(53) = \{e\}$
 $First(54) = \{d\}$

$First(55) = \{u\}$
 $First(56) = \{i\}$
 $First(57) = \{n\}$
 $First(58) = \{r\}$
 $First(59) = \{s\}$
 $First(60) = \{n\}$
 $First(61) = \{i\}$
 $First(62) = \{r\}$
 $First(63) = \{i\}$
 $First(64) = \{s\}$
 $First(65) = \{i\}$
 $First(66) = first(A) = \{i\}$
 $First(67) = first(I) = \{f\}$
 $First(68) = first(H) = \{h\}$
 $First(69) = first(W) = \{w\}$
 $First(70) = first(J) = \{j\}$
 $First(71) = first(< Llama >) = \{\}$
 $First(72) = first(< Devuelve >) = \{z\}$
 $First(73) = \{c\}$
 $First(74) = \{\}$
 $First(75) = first(P) = \{c\ i\ f\ h\ w\ j\ [z\}$
 $First(76) = \{w\}$
 $First(77) = \{f\}$
 $First(78) = \{t\}$
 $First(79) = \{\}$
 $First(80) = \{j\}$
 $First(81) = \{i\}$
 $First(82) = \{;\}$
 $First(83) = first(R) = \{i\ n\ r\ s\}$
 $First(84) = \{;\}$
 $First(85) = \{i\}$
 $First(86) = \{\}$
 $First(87) = \{h\}$
 $First(88) = \{a\}$
 $First(89) = \{\}$
 $First(90) = \{o\}$
 $First(91) = \{\}$
 $First(92) = \{q\}$
 $First(93) = \{\}$
 $First(94) = \{z\}$
 $First(95) = first(V) = \{i\ n\ r\ s\}$
 $First(96) = \{\}$

$First(97) = \{\}$
 $First(98) = \{\}$
 $First(99) = first(V) = \{i\ n\ r\ s\}$
 $First(100) = \{,\}$
 $First(101) = \{\}$

Se calculó el conjunto *first* de cada elemento no terminal:

$First(< Program >) = first(< func >) = \{b\ g\ \#\ y\ x\}$
 $First(< otraFunc >) = first(< func >) = \{b\ g\ \#\ y\ x\}$
 $First(< Func >) = first(< tipo >) = \{b\ g\ \#\ y\ x\}$
 $First(< Param >) = first(< tipo >) = \{b\ g\ \#\ y\ x\}$
 $First(< otroParam >) = \{,\}$
 $First(< Cuerpo >) = first(< decl >) \cup first(< otroParam >) = \{b\ g\ \#\ y\ x\ ,\}$
 $First(< Decl >) = first(D) = \{b\ g\ \#\ y\ x\}$
 $First(D) = first(< tipo >) = \{b\ g\ \#\ y\ x\}$
 $First(< Tipo >) = \{b\ g\ \#\ y\ x\}$
 $First(K) = \{i\}$
 $First(Q) = \{= ,\}$
 $First(N) = \{n\ r\ s\}$
 $First(C) = \{,\}$
 $First(A) = \{i\}$
 $First(A') = \{s\} \cup first(E) = \{s\ (i\ n\ r\ [\}$
 $First(E) = first(T) = \{(i\ n\ r\ [\}$
 $First(E') = \{+ -\}$
 $First(T) = first(F) = \{(i\ n\ r\ [\}$
 $First(T') = \{* / \ \% \ ^\}$
 $First(F) = \{(i\ n\ r\} \cup first(< llama >) = \{(i\ n\ r\} \cup \{\} = \{(i\ n\ r\ [\}$
 $First(R) = \{i\ n\ r\ s\}$
 $First(R') = \{> < l\ e\ d\ u\}$
 $First(V) = \{i\ n\ r\ s\}$
 $First(V') = \{n\ i\}$
 $First(V'') = \{r\ i\}$
 $First(V''') = \{s\ i\}$
 $First(P) = \{c\} \cup first(A) \cup first(I) \cup first(H) \cup first(W) \cup first(J)$
 $\quad \cup first(< llama >) \cup first(< Devuelve >) = \{c\ i\ f\ h\ w\ j\ [z\}$
 $First(< ListaP >) = first(P) = \{c\ i\ f\ h\ w\ j\ [z\}$
 $First(W) = \{w\}$
 $First(I) = \{f\}$
 $First(I') = \{t\}$
 $First(J) = \{j\}$
 $First(Y) = \{i ;\}$

$First(X) = \{;\} \cup first\ R = \{;\ i\ n\ r\ s\}$
 $First(Z) = \{i\}$
 $First(H) = \{h\}$
 $First(C') = \{a\}$
 $First(O') = \{o\}$
 $First(U) = \{q\}$
 $First(< Devuelve >) = \{z\}$
 $First(< valor >) = first(V) = \{i\ n\ r\ s\}$
 $First(< Llama >) = \{[\}$
 $First(< arg >) = first(V) = \{i\ n\ r\ s\}$
 $First(< otroArg >) = \{,\}$

Se calculó el conjunto *follow* para los no terminales *anulables* que se identificaron anteriormente.

$Follow(< otraFunc >) = follow(< Program >) \cup \{-\} = \{-|\}$
 $Follow(< Param >) = \{)\}$
 $Follow(< otroParam >) = follow(< Param >) = \{)\}$
 $Follow(< Cuerpo >) = \{\}\}$
 $Follow(< Decl >) = first(< ListaP >) \cup follow(< Cuerpo >)$
 $\quad = first(< ListaP >) \cup \{\}\} = \{c\ i\ f\ h\ w\ j\ [z\} \cup \{\}\}$
 $\quad = \{c\ i\ f\ h\ w\ j\ [z\}\}$
 $Follow(Q) = follow(K) = \{;\}$
 $Follow(C) = follow(Q) = \{;\}$
 $Follow(E') = follow(E) = \{)\}$ $\cup follow(A') = \{)\}$ $\cup \{;\} = \{)\ ;\}$
 $Follow(T') = follow(T) = first(E') \cup follow(E') \cup follow(E)$
 $\quad = \{+ -\} \cup \{)\ ;\} \cup \{)\ ;\} = \{+ -\ ;\}$
 $Follow(< ListaP >)$
 $\quad = \{\}\} \cup first(I') \cup \{;\} \cup follow(I') \cup \{\}\} \cup first(U) \cup first(C')$
 $\quad \cup follow(C') \cup follow(O') = \{t : q\ a\} \cup first(O') \cup \{\}\} \cup follow(O')$
 $\quad = \{t : q\ a\ o\}$
 $Follow(I') = \{;\}$
 $Follow(C') = first(O') \cup \{\}\} = \{o\}\}$
 $Follow(O') = \{\}\}$
 $Follow(U) = first(C') \cup follow(C') = \{a\} \cup \{o\}\} = \{a\ o\}\}$
 $Follow(< valor >) = \{)\}$
 $Follow(< arg >) = \{)\}$
 $Follow(< otroArg >) = follow(< arg >) = \{)\}$

De lo anterior, se definieron los conjuntos de selección para cada producción:

$cs(1) = \{b\ g\ \# \ y\ x\}$

$cs(2) = \{b\ g\ \#\ y\ x\}$
 $cs(3) = follow(< otraFunc >) = \{-\}$
 $cs(4) = \{b\ g\ \#\ y\ x\}$
 $cs(5) = \{b\ g\ \#\ y\ x\}$
 $cs(6) = follow(< Param >) = \{\}$
 $cs(7) = \{,\}$
 $cs(8) = follow(< otroParam >) = \{\}$
 $cs(9) = first(9) \cup follow(< Decl >)$
 $\quad = \{b\ g\ \#\ y\ x\ c\ i\ f\ h\ w\ j\ [z] \cup \{c\ i\ f\ h\ w\ j\ [z]\}$
 $\quad = \{b\ g\ \#\ y\ x\ c\ i\ f\ h\ w\ j\ [z]\}$
 $cs(10) = follow(< Decl >) = \{c\ i\ f\ h\ w\ j\ [z]\}$
 $cs(11) = \{b\ g\ \#\ y\ x\}$
 $cs(12) = \{b\ g\ \#\ y\ x\}$
 $cs(13) = \{b\}$
 $cs(14) = \{g\}$
 $cs(15) = \{\#\}$
 $cs(16) = \{y\}$
 $cs(17) = \{x\}$
 $cs(18) = \{i\}$
 $cs(19) = follow(Q) = \{;\}$
 $cs(20) = \{=\}$
 $cs(21) = \{,\}$
 $cs(22) = \{n\}$
 $cs(23) = \{r\}$
 $cs(24) = \{s\}$
 $cs(25) = follow(C) = \{;\}$
 $cs(26) = \{,\}$
 $cs(27) = \{i\}$
 $cs(28) = \{s\}$
 $cs(29) = \{(i\ n\ r\ [$
 $cs(30) = \{(i\ n\ r\ [$
 $cs(31) = \{+\}$
 $cs(32) = \{-\}$
 $cs(33) = follow(E') = \{\};\}$
 $cs(34) = \{(i\ n\ r\ [$
 $cs(35) = \{*\}$
 $cs(36) = \{/ \}$
 $cs(37) = \{\backslash\}$
 $cs(38) = \{\%\}$
 $cs(39) = \{^\wedge\}$
 $cs(40) = follow(T') = \{+ - \};\}$
 $cs(41) = \{\}$

$Cs(42) = \{i\}$
 $Cs(43) = \{n\}$
 $Cs(44) = \{r\}$
 $Cs(45) = \{\}$
 $Cs(46) = \{i\}$
 $Cs(47) = \{n\}$
 $Cs(48) = \{r\}$
 $Cs(49) = \{s\}$
 $Cs(50) = \{>\}$
 $Cs(51) = \{<\}$
 $Cs(52) = \{l\}$
 $Cs(53) = \{e\}$
 $Cs(54) = \{d\}$
 $Cs(55) = \{u\}$
 $Cs(56) = \{i\}$
 $Cs(57) = \{n\}$
 $Cs(58) = \{r\}$
 $Cs(59) = \{s\}$
 $Cs(60) = \{n\}$
 $Cs(61) = \{i\}$
 $Cs(62) = \{r\}$
 $Cs(63) = \{i\}$
 $Cs(64) = \{s\}$
 $Cs(65) = \{i\}$
 $Cs(66) = \{i\}$
 $Cs(67) = \{f\}$
 $Cs(68) = \{h\}$
 $Cs(69) = \{w\}$
 $Cs(70) = \{j\}$
 $Cs(71) = \{\}$
 $Cs(72) = \{z\}$
 $Cs(73) = \{c\}$
 $Cs(74) = follow(< ListaP >) = \{ \} t : q a o \}$
 $Cs(75) = \{c i f h w j [z\}$
 $Cs(76) = \{w\}$
 $Cs(77) = \{f\}$
 $Cs(78) = \{t\}$
 $Cs(79) = follow(I') = \{ : \}$
 $Cs(80) = \{j\}$
 $Cs(81) = \{i\}$
 $Cs(82) = \{ ; \}$
 $Cs(83) = \{i n r s\}$

$$\begin{aligned}
Cs(84) &= \{;\} \\
Cs(85) &= \{i\} \\
Cs(86) &= \{\}\} \\
Cs(87) &= \{h\} \\
Cs(88) &= \{a\} \\
Cs(89) &= follow(C') = \{o\}\} \\
Cs(90) &= \{o\} \\
Cs(91) &= follow(O') = \{\}\} \\
Cs(92) &= \{q\} \\
Cs(93) &= follow(U) = \{a o\}\} \\
Cs(94) &= \{z\} \\
Cs(95) &= \{i n r s\} \\
Cs(96) &= follow(< Valor >) = \{\}\} \\
Cs(97) &= \{\} \\
Cs(98) &= follow(< arg >) = \{\}\} \\
Cs(99) &= \{i n r s\} \\
Cs(100) &= \{;\} \\
Cs(101) &= follow(< otroArg >) = \{\}\}
\end{aligned}$$

De los conjuntos de selección obtenidos, se observó que para los mismos elementos no terminales se tienen conjuntos de selección disjuntos, por lo que se logró confirmar que se trata de una gramática LL(1). Además, a partir de estos conjuntos se definió el comportamiento de las funciones auxiliares para implementar al analizador sintáctico recursivo.

Indicaciones de ejecución

Para ejecutar el programa correctamente, en primer lugar, se debe ubicar el directorio que contiene al archivo con extensión **.l**, llamado **analizador.l**. Este archivo contiene a la implementación en Flex del analizador léxico-sintáctico. Se recomienda localizar a este archivo en una carpeta dedicada, pues la ejecución del programa genera diferentes archivos de texto. Ubicado en el directorio, se debe ejecutar el comando

```
flex analizador.l
```

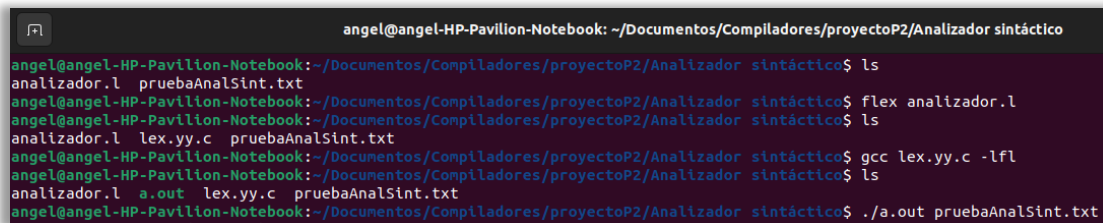
de lo cual en el directorio se generará un archivo de C, llamado "lex.yy.c". Es necesario compilar a dicho archivo para obtener al ejecutable del programa. Para compilar se utiliza el siguiente comando.

```
lex.yy.c -lfl
```

Una vez compilado el archivo de C, de la manera anterior, se obtiene un ejecutable con nombre "a.out", el cual puede ser ejecutado como cualquier programa en lenguaje C. Para obtener salidas útiles, se debe argumentar el nombre de un archivo de texto que cuyo contenido sea un programa escrito en el lenguaje definido del curso; esto se hace de la siguiente manera.

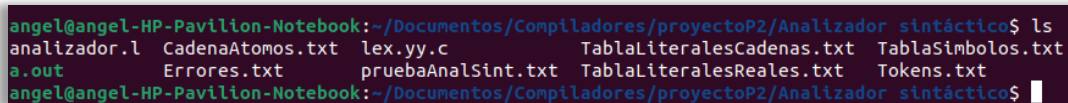
```
./a.out <nombre del programa fuente>
```

Al ejecutar el programa, de manera directa en la terminal se mostrarán varias secciones que describen información de utilidad de la compilación del programa fuente. Estas secciones son las mismas que las que se presentaron en el analizador léxico, junto con una sección adicional que refiere a la cadena de átomos generada a partir del programa fuente argumentado. Entonces, en total, se generan 6 secciones: tokens, tabla de literales reales, tabla de literales cadenas, tabla de símbolos, cadena de átomos y errores. La información mostrada en la terminal también se expresará en archivos de texto, los cuales se generarán tras cada ejecución en el directorio en el que se localice el ejecutable; dichos archivos tienen los nombres "Tokens.txt", "TablaLiteralesReales.txt", "TablaLiteralesConstantes.txt", "TablaSimbolos.txt", "CadenaAtomos.txt" y "Errores.txt".



```
angel@angel-HP-Pavilion-Notebook: ~/Documentos/Compiladores/proyectoP2/Analizador sintáctico
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ ls
analizador.l  pruebaAnalSint.txt
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ flex analizador.l
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ ls
analizador.l  lex.yy.c  pruebaAnalSint.txt
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ gcc lex.yy.c -lfl
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ ls
analizador.l  a.out  lex.yy.c  pruebaAnalSint.txt
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ ./a.out pruebaAnalSint.txt
```

Compilación y ejecución del programa analizador.l



```
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$ ls
analizador.l  CadenaAtomos.txt  lex.yy.c  TablaLiteralesCadenas.txt  TablaSimbolos.txt
a.out         Errores.txt      pruebaAnalSint.txt  TablaLiteralesReales.txt  Tokens.txt
angel@angel-HP-Pavilion-Notebook:~/Documentos/Compiladores/proyectoP2/Analizador sintáctico$
```

Archivos generados después de la ejecución del programa

Conclusiones

En general, en esta segunda entrega se pudo implementar la etapa de análisis sintáctico de un compilador para un lenguaje definido por el grupo, y también se pudo corregir y reforzar la etapa de análisis léxico que tuvo algunos errores anteriormente. Considero que esta etapa fue más complicada que la previa, pues requirió un análisis teórico más profundo y detallado. Mientras que en el análisis léxico lo complicado fue definir expresiones regulares adecuadas, en el análisis sintáctico lo complicado fue calcular a los conjuntos de selección de las producciones y eventualmente, definir a las funciones auxiliares recursivas para implementar el análisis de manera recursiva. La cantidad de

producciones planteadas para la gramática del lenguaje definido hizo que el proceso de obtención de los conjuntos de selección fuera bastante largo; además la posterior implementación de las funciones recursivas no fue más sencilla, pues para hacer esto se tuvo que hacer una constante revisión de los conjuntos de selección y la estructura de las producciones.

Esta segunda entrega requirió menos tecnicismos del lenguaje C, como prueba de esto, en el programa correspondiente únicamente se implementa una estructura de datos adicional para manipular a la cadena de átomos; no obstante, eso no significa que el presente desarrollo sea más simple, por el contrario, la gran cantidad de funciones recursivas, aunque no sean complejas en su contenido, hizo que el programa en sí fuera más difícil de mantener y corregir en caso de errores.

A pesar de las dificultades mencionadas, se pudo desarrollar un programa funcional para ejecutar a las etapas de análisis léxico y sintáctico de un compilador para el lenguaje definido, así que se cumplió con lo solicitado para esta entrega.