



UNIVERSIDAD NACIONAL AUTÓNOMA DE
MÉXICO

FACULTAD DE INGENIERÍA

Proyecto 2

Analizador Sintactico

PRESENTA

Navarrete Zamora Aldo Yael

PROFESORA

Laura Sandoval Montaña

ASIGNATURA

Compiladores

1. Objetivos

Construir, en un mismo programa, los analizadores Lexico y Sintactico Descendente Recursivo que revisen programas escritos en el lenguaje definido por la gramatica definida en la clase.

2. Descripción del problema

Un compilador se compone de tres analizadores principalmente:

1. **El analizador léxico**, que se encarga de reconocer cada caracter a la entrada, generar las tablas de símbolos, además de la tabla de cadenas, entre otros. Finalmente, manda esta información en forma de tokens y cadena de átomos al analizador sintáctico.
2. **El analizador sintáctico**, el cual se encarga de reconocer la cadena de átomos generada por el analizador léxico y detectar errores de sintaxis dentro de nuestro programa.
3. **El analizador semántico**, que utiliza la información en la tabla de símbolos para comprobar la consistencia semántica del programa fuente con la definición del lenguaje. También recopila información sobre el tipo y la guarda en la tabla de símbolos, para usarla más tarde durante la generación de código intermedio.

El analizador sintáctico será el discutido y documentado en este archivo y realizado en este segundo proyecto de la materia de compiladores. Este analizador sintáctico se desarrollará en un archivo distinto al del analizador léxico. El analizador léxico será el encargado de generar la cadena de átomos y los tokens, los cuales serán utilizados por el analizador sintáctico para realizar la verificación de la sintaxis del programa. Estos se describirán en breve debido a que primero debemos aclarar qué modificaciones se hicieron al programa del analizador léxico.

2.1. Modificaciones al analizador léxico

Para este proyecto, partimos de la salida que nos generará el analizador léxico una vez procesada la información del archivo de entrada.

Dicho analizador léxico generará las tablas de símbolos y literales, este archivo el cuál contiene al analizador léxico **importará al parser o analizador sintáctico** mediante un header creado para que este pueda hacer uso de la información que se generó en el analizador léxico tales como la tabla de literales o cadenas, la tabla de símbolos, los tokens y la cadena de átomos.

La cadena de átomos es generada por el analizador léxico en un archivo distinto llamado 'lex_an_cadena_de_atomos.out' y es la que se utilizará para realizar la verificación de la sintaxis del programa. Dentro del analizador léxico se agrega una sentencia switch case, para poder definir el átomo dependiendo la clase asociada a cada uno de los tokens. Esto es de suma importancia porque una vez definidos los tokens y la cadena de átomos el compilador puede continuar con su análisis.

2.2. Descripción del analizador sintáctico o solución del problema

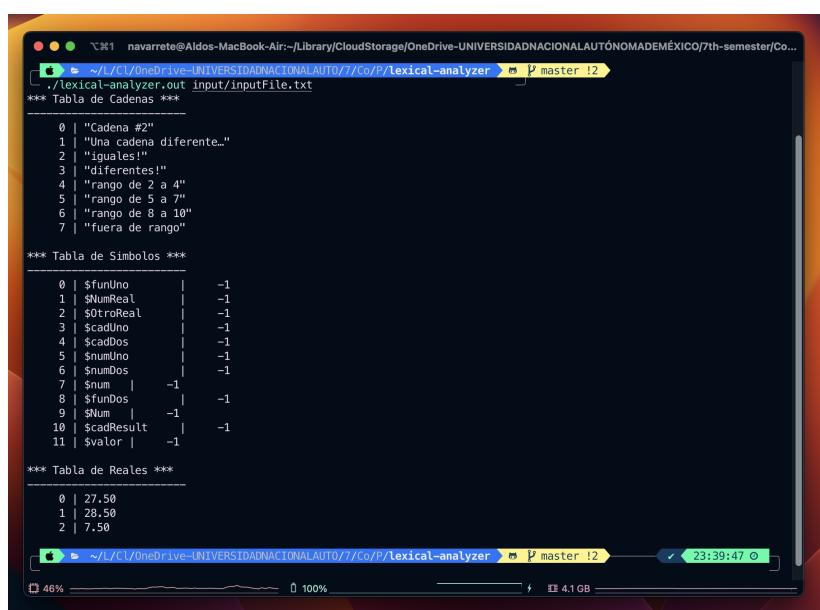
Finalmente, para solucionar el problema o realizar este analizador, definimos un archivo `.c` aparte para implementar cada una de las funciones para crear el analizador descendente recursivo, estas funciones llevan condiciones dentro de ellas para poder reconocer la sintaxis del programa y cada una de estas funciones llama recursivamente a las otras encargadas de analizar el lenguaje guiándose de nuestra gramática. Existe una función por cada no terminal aparecido en la gramática descrita más adelante en este documento. Para poder realizar este analizador, se importó el archivo `.h` que contiene las funciones del analizador léxico, una vez importado, simplemente llamamos a la función `parser()` que se encarga de llamar a las demás funciones que se encargan de reconocer la sintaxis del programa.

Esta función del parser, primeramente abrirá el archivo en el cual se guardó la cadena de átomos mediante la función de C `fopen()` y posteriormente se llamará a la función `programa()` que se encarga de reconocer la sintaxis del programa. Esta función `programa()` se encarga de llamar a las demás funciones que se encargan de reconocer la sintaxis del programa de manera **recursiva**.

En cada llamada recursiva del programa, se hacen algunas validaciones de caracteres tal y como se enseñó en teoría el funcionamiento del analizador, en caso de que el archivo de entrada tuviese algún error sintáctico, el programa se detendrá y se mostrará un mensaje de error. Este error nos mencionará en dónde está ubicado el error, es decir, en qué átomo fué encontrado el error y qué caracter se esperaba en ese átomo.

En caso de que el programa no tuviese ningún error, se mostrará un mensaje de éxito 'SYNTAX SUCCESS' indicando que el programa no tuvo ningún error sintáctico y cerrando el archivo de entrada.

Finalmente, en consola observamos una salida con cada una de las tablas asociadas al analizador sintáctico.



```
./lexical-analyzer.out input/inputFile.txt
*** Tabla de Cadenas ***
0 | "Cadena #2"
1 | "Una cadena diferente."
2 | "Iguales!"
3 | "diferentes!"
4 | "rango de 2 a 4"
5 | "rango de 5 a 7"
6 | "rango de 8 a 10"
7 | "fuera de rango"

*** Tabla de Símbolos ***
0 | $funcion | -1
1 | $numReal | -1
2 | $otroReal | -1
3 | $cadUno | -1
4 | $cadDos | -1
5 | $numUno | -1
6 | $numDos | -1
7 | $num | -1
8 | $funcion | -1
9 | $num | -1
10 | $cadResult | -1
11 | $valor | -1

*** Tabla de Reales ***
0 | 27.50
1 | 28.50
2 | 7.50
```

Figura 1: Salida en consola del analizador léxico.

2.3. Descripción del lenguaje o gramática

El lenguaje que describe la gramática se enlistará mediante las producciones de los elementos no terminales de la siguiente tabla.

1:	$\langle \text{Program} \rangle \rightarrow \langle \text{Func} \rangle \langle \text{otraFunc} \rangle$	42:	$F \rightarrow i$
2:	$\langle \text{otraFunc} \rangle \rightarrow \langle \text{Func} \rangle \langle \text{otraFunc} \rangle$	43:	$F \rightarrow n$
3:	$\langle \text{otraFunc} \rangle \rightarrow \xi$	44:	$F \rightarrow r$
4:	$\langle \text{Func} \rangle \rightarrow \langle \text{Tipo} \rangle i \langle \text{Param} \rangle \{ \langle \text{Cuerpo} \rangle \}$	45:	$F \rightarrow \langle \text{Llama} \rangle$
5:	$\langle \text{Param} \rangle \rightarrow \langle \text{Tipo} \rangle i \langle \text{otroParam} \rangle$	46:	$R \rightarrow iR'V$
6:	$\langle \text{Param} \rangle \rightarrow \xi$	47:	$R \rightarrow nR'V'$
7:	$\langle \text{otroParam} \rangle \rightarrow \langle \text{Tipo} \rangle i \langle \text{otroParam} \rangle$	48:	$R \rightarrow rR'V''$
8:	$\langle \text{otroParam} \rangle \rightarrow \xi$	49:	$R \rightarrow sR'V'''$
9:	$\langle \text{Cuerpo} \rangle \rightarrow \langle \text{Decl} \rangle \langle \text{listaP} \rangle$	50:	$R' \rightarrow >$
10:	$\langle \text{Decl} \rangle \rightarrow \xi$	51:	$R' \rightarrow <$
11:	$\langle \text{Decl} \rangle \rightarrow D \langle \text{Decl} \rangle$	52:	$R' \rightarrow l$
12:	$D \rightarrow \langle \text{Tipo} \rangle K;$	53:	$R' \rightarrow e$
13:	$\langle \text{Tipo} \rangle \rightarrow b$	54:	$R' \rightarrow d$
14:	$\langle \text{Tipo} \rangle \rightarrow g$	55:	$R' \rightarrow u$
15:	$\langle \text{Tipo} \rangle \rightarrow \#$	56:	$V \rightarrow i$
16:	$\langle \text{Tipo} \rangle \rightarrow y$	57:	$V \rightarrow n$
17:	$\langle \text{Tipo} \rangle \rightarrow x$	58:	$V \rightarrow r$
18:	$K \rightarrow iQ$	59:	$V \rightarrow s$
19:	$Q \rightarrow \xi$	60:	$V' \rightarrow n$
20:	$Q \rightarrow =NC$	61:	$V' \rightarrow i$
21:	$Q \rightarrow ,K$	62:	$V'' \rightarrow r$
22:	$N \rightarrow n$	63:	$V'' \rightarrow i$
23:	$N \rightarrow r$	64:	$V''' \rightarrow s$
24:	$N \rightarrow s$	65:	$V''' \rightarrow i$
25:	$C \rightarrow \xi$	66:	$P \rightarrow A$
26:	$C \rightarrow ,K$	67:	$P \rightarrow I$
27:	$A \rightarrow i=A';$	68:	$P \rightarrow H$
28:	$A' \rightarrow s$	69:	$P \rightarrow W$
29:	$A' \rightarrow E$	70:	$P \rightarrow J$
30:	$E \rightarrow T E'$	71:	$P \rightarrow \langle \text{Llama} \rangle$
31:	$E' \rightarrow + T E'$	72:	$P \rightarrow \langle \text{Devuelve} \rangle$
32:	$E' \rightarrow - T E'$	73:	$P \rightarrow c;$
33:	$E' \rightarrow \xi$	74:	$\langle \text{listaP} \rangle \rightarrow \xi$
34:	$T \rightarrow F T'$	75:	$\langle \text{listaP} \rangle \rightarrow P \langle \text{listaP} \rangle$
35:	$T' \rightarrow * F T'$	76:	$W \rightarrow w(R)m\{\langle \text{listaP} \rangle\}$
36:	$T' \rightarrow / F T'$	77:	$I \rightarrow f(R)\langle \text{listaP} \rangle I'$
37:	$T' \rightarrow \backslash F T'$	78:	$I' \rightarrow t \langle \text{listaP} \rangle$
38:	$T' \rightarrow \% F T'$	79:	$I' \rightarrow \xi$
39:	$T' \rightarrow ^ F T'$	80:	$J \rightarrow j(YXZ\{\langle \text{listaP} \rangle\})$
40:	$T' \rightarrow \xi$	81:	$Y \rightarrow i=E;$
41:	$F \rightarrow (E)$	82:	$Y \rightarrow ;$

Figura 2: Tabla descriptiva del lenguaje (Gramática del lenguaje) definido en clase.

83:	$X \rightarrow R;$	93:	$U \rightarrow \xi$
84:	$X \rightarrow ;$	94:	$\langle \text{Devuelve} \rangle \rightarrow z(\langle \text{valor} \rangle);$
85:	$Z \rightarrow i=E)$	95:	$\langle \text{valor} \rangle \rightarrow V$
86:	$Z \rightarrow)$	96:	$\langle \text{valor} \rangle \rightarrow \xi$
87:	$H \rightarrow h(i)\{C'O'\}$	97:	$\langle \text{Llama} \rangle \rightarrow [i(\langle \text{arg} \rangle)]$
88:	$C' \rightarrow \text{an}:\langle \text{listaP} \rangle UC'$	98:	$\langle \text{arg} \rangle \rightarrow \xi$
89:	$C' \rightarrow \xi$	99:	$\langle \text{arg} \rangle \rightarrow V \langle \text{otroArg} \rangle$
90:	$O' \rightarrow o:\langle \text{listaP} \rangle$	100:	$\langle \text{otroArg} \rangle \rightarrow ,V \langle \text{otroArg} \rangle$
91:	$O' \rightarrow \xi$	101:	$\langle \text{otroArg} \rangle \rightarrow \xi$
92:	$U \rightarrow q$		

Figura 3: Tabla descriptiva del lenguaje (Gramática del lenguaje) definido en clase.

3. Conjuntos de selección para cada producción

Conjuntos de selección para cada producción	
$c.s(1) = \{bgyx\}$	$c.s(51) = \{<\}$
$c.s(2) = \{bgyx\}$	$c.s(52) = \{l\}$
$c.s(3) = \{- \}$	$c.s(53) = \{e\}$
$c.s(4) = \{bgyx\}$	$c.s(54) = \{d\}$
$c.s(5) = \{bgyx\}$	$c.s(55) = \{u\}$
$c.s(6) = \{)\}$	$c.s(56) = \{i\}$
$c.s(7) = \{,\}$	$c.s(57) = \{n\}$
$c.s(8) = \{)\}$	$c.s(58) = \{r\}$
$c.s(9) = \{bgxyifhwj[zc\}$	$c.s(59) = \{s\}$
$c.s(10) = \{ifhwj[zc\}$	$c.s(60) = \{n\}$
$c.s(11) = \{bgyx\}$	$c.s(61) = \{i\}$
$c.s(12) = \{bgyx\}$	$c.s(62) = \{r\}$
$c.s(13) = \{b\}$	$c.s(63) = \{i\}$
$c.s(14) = \{g\}$	$c.s(64) = \{s\}$
$c.s(15) = \{\}$	$c.s(65) = \{i\}$
$c.s(16) = \{y\}$	$c.s(66) = \{i\}$
$c.s(17) = \{x\}$	$c.s(67) = \{f\}$
$c.s(18) = \{i\}$	$c.s(68) = \{h\}$
$c.s(19) = \{;\}$	$c.s(69) = \{w\}$
$c.s(20) = \{=\}$	$c.s(70) = \{j\}$
$c.s(21) = \{,\}$	$c.s(71) = \{\}$
$c.s(22) = \{n\}$	$c.s(72) = \{z\}$
$c.s(23) = \{r\}$	$c.s(73) = \{c\}$
$c.s(24) = \{s\}$	$c.s(74) = \{t : qao\}$
$c.s(25) = \{;\}$	$c.s(75) = \{ifhwj[zc\}$
$c.s(26) = \{,\}$	$c.s(76) = \{w\}$
$c.s(27) = \{i\}$	$c.s(77) = \{f\}$
$c.s(28) = \{s\}$	$c.s(78) = \{t\}$
$c.s(29) = \{(inr[$	$c.s(79) = \{:\}$

$c.s(30) = \{(inr[]\}$	$c.s(80) = \{j\}$
$c.s(31) = \{+\}$	$c.s(81) = \{i\}$
$c.s(32) = \{-\}$	$c.s(82) = \{;\}$
$c.s(33) = \{;\}$	$c.s(83) = \{inrs\}$
$c.s(34) = \{(inr[]\}$	$c.s(84) = \{;\}$
$c.s(35) = \{*\}$	$c.s(85) = \{i\}$
$c.s(36) = \{/ \}$	$c.s(86) = \{\}$
$c.s(37) = \{\}$	$c.s(87) = \{h\}$
$c.s(38) = \{\%\}$	$c.s(88) = \{a\}$
$c.s(39) = \{^{\circ}\}$	$c.s(89) = \{o\}$
$c.s(40) = \{+-;)\}$	$c.s(90) = \{o\}$
$c.s(41) = \{(\}$	$c.s(91) = \{\}$
$c.s(42) = \{i\}$	$c.s(92) = \{q\}$
$c.s(43) = \{n\}$	$c.s(93) = \{ao\}$
$c.s(44) = \{r\}$	$c.s(94) = \{z\}$
$c.s(45) = \{[]\}$	$c.s(95) = \{inrs\}$
$c.s(46) = \{i\}$	$c.s(96) = \{\}$
$c.s(47) = \{n\}$	$c.s(97) = \{[]\}$
$c.s(48) = \{r\}$	$c.s(98) = \{\}$
$c.s(49) = \{s\}$	$c.s(99) = \{inrs\}$
$c.s(50) = \{>\}$	$c.s(100) = \{,\}$
	$c.s(101) = \{\}$

4. Ejecución del programa

El programa se encuentra comentado, con una descripción breve de lo que hace cada una de las funciones. La sangría está muy cuidada y se ve bastante presentable y legible el código. Para correr el programa, debemos tener instalado flex en nuestro equipo y seguir los siguientes pasos una vez que lo tengamos instalado.

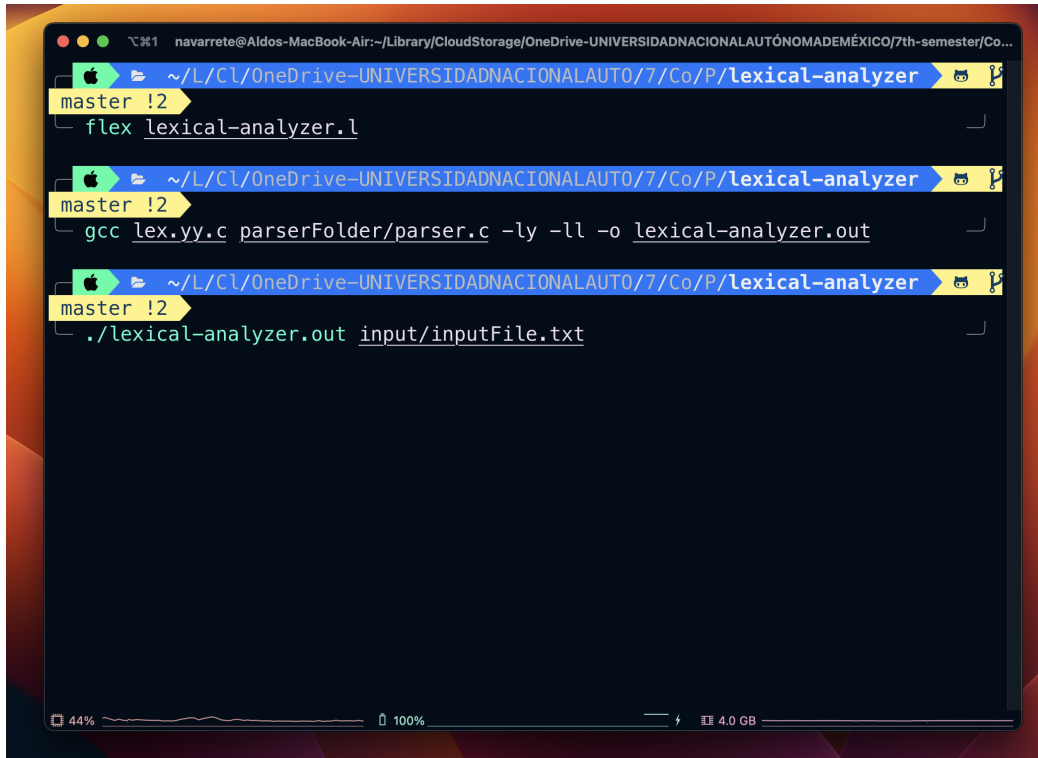
1. Abrir en una ventana de terminal la carpeta donde se encuentran todos los archivos.
2. Una vez dentro de la carpeta correremos el comando `flex lexical-analyzer.l`
3. `gcc lex.yy.c -ly -ll -o lexical-analyzer.out` para compilarlo mediante gcc y tener un archivo de salida ejecutable para un entorno LINUX.
4. Finalmente, `./lexical-analyzer.out input/inputFile.txt` es importante destacar que, el archivo *inputFile.txt* es el archivo de entrada, **en caso de no teclear ningún archivo de entrada el programa puede manejar este tipo de problema mandando un mensaje de que es necesaria esa entrada por consola.**

Una vez ejecutados los pasos anteriores, podremos observar **en la carpeta output algunos archivos de salida:**

1. El archivo de texto para la generación de los tokens.
2. El archivo de texto para la la impresión de errores léxicos.

3. El archivo de salida del parser, el cual nos dirá si el archivo de entrada es aceptado o no por el lenguaje mediante la evaluación del analizador sintáctico descendente recursivo.
4. El archivo de cadena de átomos, el cual nos dirá la cadena de átomos que se generó a partir del archivo de entrada y del cuál obtendrá entrada el analizador sintáctico.

A continuación represento la ejecución de dichos comandos en la terminal.



```

navarrete@Aldos-MacBook-Air:~/Library/CloudStorage/OneDrive-UNIVERSIDADNACIONALAUTONOMADEMEXICO/7th-semester/Co...
~ /L/CL/OneDrive-UNIVERSIDADNACIONALAUT0/7/Co/P/lexical-analyzer
master !2
flex lexical-analyzer.l

~ /L/CL/OneDrive-UNIVERSIDADNACIONALAUT0/7/Co/P/lexical-analyzer
master !2
gcc lex.yy.c parserFolder/parser.c -ly -ll -o lexical-analyzer.out

~ /L/CL/OneDrive-UNIVERSIDADNACIONALAUT0/7/Co/P/lexical-analyzer
master !2
./lexical-analyzer.out input/inputFile.txt
  
```

Figura 4: Tabla descriptiva de átomo para cada palabra reservada en nuestro lenguaje.

5. Conclusiones

“ En el proyecto realizado esta vez, el analizador sintáctico leyó correctamente algunos archivos personales de entrada así como el archivo proporcionado por la profesora Laura a través de la plataforma de Chamilo, dentro del analizador léxico se genera la cadena de átomos, la cual fue un requisito para este proyecto debido a que este analizador realizado necesitaba o hacía uso de dicha cadena de átomos.

Tuve complicaciones al momento de abrir un archivo para lectura en el parser o analizador sintáctico, debido a que en el analizador léxico ya lo había abierto para su lectura, entonces me vi en la necesidad de buscar documentación oficial para ver cómo podía solucionar el problema de volver a poner el apuntador del buffer al inicio de dicho archivo.

Concluyo que el analizador sintáctico fue desarrollado de forma correcta en este proyecto, cumple con las características comentadas por la profesora, y además, fueron corregidos algunos errores/observaciones hechas por la profesora para el analizador léxico, entre ellas, la lectura de las cadenas. Me pareció muy buena implementación la escritura en otro archivo de salida para la cadena de átomos, esto resulta muy fácil debido a que cada salida está modularizada. “

— **Navarrete Zamora Aldo Yael**, Estudiante de Ingeniería en Computación