UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Proyecto 2

Analizador Síntactico

PRESENTA

Navarrete Zamora Aldo Yael

PROFESORA

Laura Sandoval Montaño

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.

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, así como los tokens que se transferirán al analizador sintáctico. Este analizador léxico será realizado en lex, LEX es un generador de programas diseñado para procesamiento léxico de flujo de entrada y salida de caracteres. Pero, este analizador tiene algunas restricciones, como por ejemplo:

En este documento se describe el problema para afrontar dentro del desarrollo del software que corresponde al analizador sintáctico.

El número de clase es **inamovible**.

Además, el analizador léxico tendrá como entrada un archivo con el programa fuente, el cual se indicará desde la línea de comandos al momento de mandar a ejecutar el analizador léxico.

Como delimitador de un componente léxico será uno o vario

3. Conjuntos de selección para cada producción

Valor	Palabra reservada	Equivale en C
0	alternative	case
1	big	long
2	evaluate	if
3	instead	else
4	large	double
5	loop	while
6	make	do
7	number	int
8	other	default
9	real	float
10	repeat	for
11	select	switch
12	small	short
13	step	continue
14	stop	break
15	symbol	char
16	throw	return

Valor	Op. relacional
0	<
1	>
2	<=
3	>=
4	==
5	!=

Figura 1: Catálogos para las palabras reservadas y los operadores relacionales.

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

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) = {}	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 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, podrémos observar **dos archivos de texto para la salida:**

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

Este último archivo reportará qué expresiones regulares no están definidas por el lenguaje desarrollado.

5. Conclusiones

"Hola

"

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