UNIVERSIDAD
RAFAEL LANDÍVAR

FACULTAD DE
INGENIERIA

Proyecto

PABLO ISAAC GARCIA ORELLANA

DANIEL ESTUARDO MOLINA CASTILLO

JENIFER ALEJANDRA RABANALES GAMBOA

AXEL CESAR OSMUNDO RODAS SAMAYOA

Carné: 15352-18

15576-18

16008-18

Introducción

Como parte del proceso de aprendizaje del curso nos encontramos con un proyecto, en los cuales debemos de aplicar los conocimientos que se han estado adquiriendo durante el ciclo estudiantil, y en este proyecto, buscamos aplicar el software de Jflex, la aplicación de tokens , arboles sintácticos y semánticos, entro muchas otras cosas.

La aplicación de este proyecto se concentró en un analizador léxico, se trabajó mediante el lenguaje de programación de Java, en la cual se evaluó los procesos y errores tanto léxicos como sintácticos y semánticos, para que de esta forma los árboles tengan una correcta aplicación dentro del programa.

Objetivo General

Poder llevar a la práctica todos los conocimientos adquiridos durante el presente ciclo, a través de la creación de un analizador léxico, mediante la implementación de Jflex y Java.

Objetivos Específicos

- Generar una programación adecuada y acorde a las necesidades del programa, sin dejar de lado la indentación y la documentación.

- Lograr generar las distintas conversiones entre todos los tipos.

- Conseguir un funcionamiento correcto de un programa completo utilizando el lenguaje LOOP.

Token	Tipo	Expresion Regular
sen	mat_funcsen	({ Math.sen{D* | identificador}) {lexeme=yytext(); return Identificador;}
cos	mat_funccos	({ Math.cos{D* | identificador}) {lexeme=yytext(); return Identificador;}
tan	mat_functan	({ Math.tan{D* | identificador}) {lexeme=yytext(); return Identificador;}
log	mat_funclog	({ Math.log{D* | identificador}) {lexeme=yytext(); return Identificador;}
sqrt	mat_funcsqrt	({ Math.sqrt{D* | identificador}) {lexeme=yytext(); return Identificador;}
fun_pre	parseInt	
fun_pre	valueOf	
byte_literal	byte	
float_literal	float	([0-9][0-9]*)(.)([0-9][0-9]*)
double_literal	double	([0-9][0-9]*)(.)([0-9][0-9])
string_literal	string	[a-zA-Z][a-zA-Z]*
chat_literal	char	[a-zA-Z]
int_literal	int	([0-9][0-9]*)([0-9][0-9]*)
long_literal	long	
short_literal	short	
boo_literal	boolean	[True]|[False]
identifier	count	
identifier	class	
identifier	println	
identifier	main	
identifier	in	
identifier	out	
identifier	System	
identifier	print	
identifier	args	
identifier	index	
package	C = {a,b,c,d,e,g,h,l,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}	([a-z][a-z]*(.)[a-z][a-z]*)*
	$paquete = U_{i\geq 0}\left((Caracter\ U_{i\geq 0} Caracter^{i})(.)(Caracter\ U_{i\geq 0} Caracter^{i})\right)^{i}$	

variableName	D = {0,1,2,3,4,5,6,7,8,9}	[a-zA-z]|[a-zA-z]([a-zA-z]*|[0-9]*)
	C = {a,b,c,d,e,g,h,l,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}	
	$variable = (Caracter\ U_{i\geq 0} Caracter^i)(Digito\ U_{i\geq 0} Digito^i)$	
className	D = {0,1,2,3,4,5,6,7,8,9}	[A-Z]([A-Za-z]*|[0-9]*)
	C = {a,b,c,d,e,g,h,l,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,}	
	$clase = (CaracterMayus\ U_{i\geq 0}\ Caracter^i)(Digito\ U_{i\geq 0} Digito^i)$	
methodName	D = {0,1,2,3,4,5,6,7,8,9}	[a-z]([A-Za-z]*[0-9]*)
	C = {a,b,c,d,e,g,h,l,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,}	
	$metodo = (CaracterMin\ U_{i\geq 0}\ Caracter^i)(Digito\ U_{i\geq 0} Digito^i)$	
sum_op	+	[NUM] (sum_op) [NUM]
res_op	-	[NUM] (res_op) [NUM]
multi_op	*	[NUM] (multi) [NUM]
div_op	/	[NUM] (div) [NUM]
mod_op	%	[NUM] (mod) [NUM]
equal_asign	=	
mul_asig	*=	
div_asig	/=	
mod_asig	%=	
and_asig	&=	
or_asig	|=	
expo_asig	^=	
rbit_asig	>>=	
lbit_asig	<<=	
dif_op	!=	
gre_op	>	
sma_op	<	
gequal_op	>=	
sequal_op	<=	
and_logop	&&	

or_logop	||	
dif_logop	!	
line_com	//	
op_com	/*	
cl_com	*/	
end_line	;	
coma	,	
point	.	
new_line	\n	
carriage_return	\r	
tab	\t	
backspace	\b	
form_feed	\f	
NUM	D = {0,1,2,3,4,5,6,7,8,9}	[0-9][0-9]*
	$Numero = Digito\ U_{i\geq 0} Digito^{i}$	
CADENA	L = {a,b,c,d,e,g,h,l,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}	[A-Za-z][a-zA-Z]*
	$Cadena = Letra\ U_{i\geq 0} Letra^{i}$	
PALABRAS RESERVADAS		
Token	Descripcion	Lexema
abstract	caracter a,b,s,t,r,a,c,t	abstract
boolean	caracter b,o,o,l,e,a,n	boolean
break	caracter b,r,e,a,k	break
catch	caracter c,a,t,c,h	catch
class	caracter c,l,a,s,s	class
const	caracter c,o,n,s,t	const
default	caracter d,e,f,a,u,l,t	default
do	caracter d,o	do

else	caracter e,l,s,e	else
extends	caracter e,x,t,e,n,d,s	extends
for	caracter f,o,r	for
goto	caracter g,o,t,o	goto
if	caracter I,f	if
implements	caracter I,m,p,l,e,m,e,n,t,s	implements
import	caracter I,m,p,o,r,t	import
interface	caracter I,n,t,e,r,f,a,c,e	interface
public	caracter p,u,b,l,I,c	public
static	caracter s,t,a,t,I,c	static
return	caracter r,e,t,u,r,n	return
throw	caracter t,h,r,o,w	throw
this	caracter t,h,I,s	this
try	caracter t,r,y	try
package	caracter p,a,c,k,a,g,e	package
private	caracter p,r,I,v,a,t,e	private
switch	caracter s,w,I,t,c,h	switch
void	caracter v,o,I,d,	void
date	caracter d,a,t,e	date
while	caracter w,h,I,l,e	while
else if	caracter e,l,s,e,I,f	else if
switch	caracter s,w,I,t,c,h	switch
for	caracter f,o,r	for
case	caracter c,a,s,e	case

Gramaticas Libres de Contexto

A continuación se desarrollara la gramática libre de contexto basada en el lenguaje de programación que se utilizo para realizar el siguiente proyecto, dicho lenguaje es Java, tomando en cuenta que los tokens fueron proporcionados en una tabla aparte.

No Terminales: {Variable, Atributo, Signo, Propiedad, Parametro, ValorInicial, Elemento, Funcion, Metodo, Hex}

Terminales: {ENTRERO, CADENA, TEXTO, PComa, DosP, px, NUMERAL, NombrePropiedad, Color, MAS, MENOS, PropiedadVariable}

Símbolo Inicial

Producciones:

Variable → TipoDato NombreVariable IGUAL ValorInicial PComa,

| NombreVariable IGUAL ValorInicial PComa,

| NombreVariable IGUAL Cero PComa

Atributo → NombreAtributo Coma | NombreAtributo

Signo → MAS
| MENOS

Propiedad → NombrePropiedad PComa
| NombrePropiedad Coma
| NombrePropiedad

Parametro → Cadena
| Color
| Hex
| ENTERO
| PComa

Return → return Parametro ConcatCadenas
| return Parametro
| return ConcatCadenas

ConcatCadenas → TEXTO Signo ConcatCadenas
| ID Signo ConcatCadenas
| NUM Signo
| Signo NUM

| TEXTO
| ID
| Signo
| NUM

ValorInicial → Cadena, Coma
|Cadena
| TEXTO Com
| TEXTO
| Entero Com
|ENTERO
| ConcatCadena
| Hex

Elemento → Propiedad Variable PComa
|Propiedad Cadena Pcoma
|Propiedad (Hex + px) PComa
|Propiedad Hex
|Propiedad Variable
|Propiedad Cadena
|Elemento Pcoma
| NombreElemento

Funcion → (NombreFuncion) CorO Elemento CorC
|CorO Elemento
| Elemento
| Elemento CorC

Metodo → NombreMetodo ParenI Propiedad ParenD DosPuntos

| Propiedad ParenI ParenD DosPuntos

| Parametro IGUAL Parametro PComa

| return ConcatCadenas PComa

Clase → NombreClase

| NombreClase OpCor Propiedad ClCor

| Propiedad

| Metodo ParenI Parametro ParanD DosPuntos

Conclusiones

El refuerzo del aprendizaje mediante ejercicios prácticos siempre tiene un impacto muy positivo, y esta no fue la excepción, y es que al poder aplicar todos los conocimientos adquiridos ya con un proyecto sirve, y sirve demasiado, puesto que gracias a esto nos damos cuenta en que estamos fallando y por ende que es lo que necesitamos reforzar.

A pesar de la complejidad con la que nos encontramos por parte del proyecto, este puedo ser llevado a cabo en su totalidad. Después de esta experiencia como grupo se puede decir que es algo más que necesario pues es la única forma en la que podemos comprender de total manera el funcionamiento de los compiladores.