Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Ingeniería en Ciencias y Sistemas
Lab. Organización de lenguajes y compiladores

GRAMATICA

Justin Josue Aguirre Román
202004734

EXPRESIONES REGULARES

Para el desarrollo correcto de la aplicación y su análisis de mensaje de entrada se desarrollaron ciertas expresiones regulares para mejor manejo y delimitación de ciertos tokens

TOKEN	EXPRESION REGULAR
COMENTARIO	**"//"[^\n]***
COMENTARIO MULTILINEA	**[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]**
CADENAS	**\'[^\']?\'**
CHAR	**\"[^\"]*\"**
DECIMAL	**[0-9]+"."[0-9]+**
ENTERO	**[0-9]+**
ID	**([a-zA-ZñÑ])[a-zA-ZñÑ0-9_]***

TERMINALES

Para este apartado se tomaron en cuenta todas aquellos símbolos y palabras reservadas, o mas bien todo aquello que no genera una producción de forma directa dentro de la gramática.

No.	Lexema	Token
1	,	Coma
2	If	if
3	else	else
4	while	while
5	for	for
6	do	do
7	return	return
8	break	break
9	continue	continue
10	Void	void

11	switch	switch
12	case	case
13	default	default
14	print	print
15	println	println
16	toLowe	toLower
17	toUpper	toUpper
18	round	round
19	length	length
20	typeof	typeOf
21	tostring	ToString
22	toCharArray	ToCharArrat
23	run	run
24	new	new
25	int	int
26	double	double
27	char	char
28	boolean	boolean
29	string	string
30	true	true
31	false	false
32	:	DosPuntos
33	;	PuntoComa
34	(	ParentesisA
35	)	ParentesisC
36		

37	[	CorcheteA
38	]	CorcheteC
39	{	LlaveA
40	}	LlaveC
41	+	mas
42	*	multiplicación
43	/	division
44	^	potencia
45	%	modulo
46	-	menos
47	==	igualIf
48	!=	diferente
49	<=	menorQue
50	>=	mayorQue
51	=	igual
52	<	menor
53	>	mayor
54	?	InterrogacionC
55	||	or
56	&&	and
57	!	not

NO TERMINALES

No	NOMBRE
1	INIT
2	INSTRUCCIÓN

3	INSTRUCCIONES
4	DECLARACION
5	ASIGNACION
6	PRINT
7	PRINT_LN
8	DECLARACION_VACIO
9	CASTEO_O
10	INCREMENTO_DECREMENTO
11	IF
12	SWITCH
13	WHILE
14	DO_WHILE
15	FOR
16	BREAK
17	CONTINUE
18	RETURN
19	LLAMADA
20	METODO_FUNCION
21	RUN
22	PARAMETROS
23	DECLARACION_VACIO
24	PARAMETROS_LLAMADA
25	I_SWITCH
26	INS_SWITCH
27	PARAMETRO1
28	PARAMETRO2
29	LISTA_FILAS
30	LISTA_VALORES
31	ID_DECLARACION
32	ELSE

33	TIPO_SW
34	DEFAULT
35	TIPO_DATO
36	EXPRESION
37	VALOR
38	GET_VALOR
39	ROUND
40	LENGTH
41	VALOR_L
42	TYPEOFF
43	TO_STRING
44	TO_UPPER
45	TO_LOWER

EXPLICACION GRAMATICA

Las semánticas y estructuras del mensaje son:

COMENTARIO: Los comentarios son una forma elegante de indicar que función tiene cierta sección del código que se ha escrito simplemente para dejar algún mensaje en específico.

Estructura:

// Este es un comentario de una línea

/* Este es un comentario

Multilínea Para este lenguaje */

DECLARACION: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador.

Estructura:

<Tipo> id;

<Tipo> id1, id2, id3, id4;

<Tipo> identificador = **<Expresión>**;

<Tipo> id1, id2, id3, id4 = **<Expresión>**;

ASIGNACION: Una variable deberá de ser declarada antes de poder ser utilizada. Todas las variables tendrán un tipo de dato y un nombre de identificador.

Estructura:

identificador = **<Expresión>**;

id1, id2, id3, id4 = **<Expresión>**;

CASTEOS: Los casteos son una forma de indicar al lenguaje que convierta un tipo de dato en otro, por lo que, si queremos cambiar un valor a otro tipo, es la forma adecuada de hacerlo.

Estructura:

(**<Tipo>**) **<Expresión>**;

INCREMENTO: si incrementamos una variable, se incrementará de uno en uno su valor.

DECREMENTO: si decrementamos una variable, se disminuirá de uno en uno su valor.

Estructura:

<Expresion>'+''+'';

<Expresion>' '-''-'';'

VECTORES: Los vectores son una estructura de datos de tamaño fijo que pueden almacenar valores de forma limitada, y los valores que pueden almacenar son de un único tipo; int, double, boolean, char o string.

Estructura:

DECLARACION TIPO 1 (una dimensión)

<Tipo> 'Id' '[' ']' = new **<Tipo>** '[' **<Expresion>** ']' ';'

<Tipo> 'Id' '['']''['']'= new **<Tipo>**'[' **<Expresion>** ']''['**<Expresion>**']' ';'

DECLARACION TIPO 2

<Tipo> 'Id' '[' ']' = '[' **<LISTAVALORES>** ']' ';'

VECTORES: La sentencia if ejecuta las instrucciones sólo si se cumple una condición. Si la condición es falsa, se omiten las sentencias dentro de la sentencia.

Estructura:

‘if’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’

| ‘if’ ‘(’**<Expresion>**‘)’‘{‘**<Instrucciones>**‘}’‘else’‘{‘ **<Instrucciones>** ‘}’

| ‘if’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’ ‘else’ **<IF>**

SWITCH: Estructura principal del switch, donde se indica la expresión a evaluar.

Estructura:

‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<CaseList> <Default>** ‘}’

| ‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<CaseList>** ‘}’

| ‘switch’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Default>** ‘}’

CASE: Estructura que contiene las diversas opciones a evaluar con la expresión establecida en el switch.

Estructura:

‘case’ **<Expresion>** ‘:’ **<Instrucciones>**

WHILE: El ciclo o bucle While, es una sentencia que ejecuta una secuencia de instrucciones mientras la condición de ejecución se mantenga verdadera.

Estructura:

‘while’ ‘(’ **<Expresion>** ‘)’ ‘{‘ **<Instrucciones>** ‘}’

FOR El ciclo o bucle for, es una sentencia que nos permite ejecutar N cantidad de veces la secuencia de instrucciones que se encuentra dentro de ella.

Estructura:

‘for‘(’<Declaracion>|<Asignacion>’;’<Condicion>’;’<Actualizacion>‘)’‘{‘<Instruccion> ‘}’

DO_WHILE: El ciclo o bucle Do-While, es una sentencia que ejecuta al menos una vez el conjunto de instrucciones que se encuentran dentro de ella y que se sigue ejecutando mientras la condición sea verdadera

Estructura:

‘do’ ‘{‘ <Instrucciones> ‘}’ ‘while’ ‘(’ <Expresion> ‘)’ ‘;’

BREAK: La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutara y este se saldrá del ciclo.

Estructura:

‘break**’ ;**

CONTINUE: La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error.

Estructura:

‘continue**’ ;**

RETURN: La sentencia return finaliza la ejecución de un método o función y puede especificar un valor para ser devuelto a quien llama a la función.

Estructura:

‘return**’ ;**

| **‘**return**’ <Expresion> ;**

FUNCIONES: Una función es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros. Para las funciones es obligatorio que las mismas posean un valor de retorno que coincida con el tipo con el que se declaró la función.

Estructura:

‘Id’ ‘(‘ **<Parametros>** ‘)’ ’:’ <Tipo> ‘{‘ **<Instrucciones>** ‘}’

Parametros -> **<Parametros>** , **<Tipo>** ‘id’

| **<Tipo>** ‘id’

METODOS: Un método también es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros, aunque a diferencia de las funciones estas subrutinas no deben de retornar un valor.

Estructura:

‘Id’ ‘(‘ **<Parametros>** ‘)’ ’:’ ‘void’ ‘{‘ **<Instrucciones>** ‘}’

Parametros -> **<Parametros>** , **<Tipo>** ‘id’

| **<Tipo>** ‘id’

LLAMADA: La llamada a una función específica la relación entre los parámetros reales y los formales y ejecuta la función. Los parámetros se asocian normalmente por posición, aunque, opcionalmente, también se pueden asociar por nombre. Si la función tiene parámetros formales por omisión, no es necesario asociarles un parámetro real.

Estructura:

LLAMADA ->'Id' '(' **<Parametros>** ')'
|'Id' '(' ')'
Parametros -> **<Parametros>** , 'id'
| 'id'

PRINT: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter. Esta función no concatena un salto de línea al final del contenido que recibe como parámetro.

Estructura:

'Print' '(' <Expresion>')';

PRINTLN: Esta función nos permite imprimir expresiones con valores únicamente de tipo entero,doble, booleano, cadena y carácter. Esta función concatena un salto de línea al finalizar el contenido que recibe como parámetro.

Estructura:

'Println' '(' <Expresion>')';

TO_LOWER: Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas.

Estructura:

'toLower' '(' <Expresion>')';

TO_UPPER: Esta función recibe como parámetro una expresión de tipo cadena retorna una nueva cadena con todas las letras mayúsculas.

Estructura:

'toUpper' '(' <Expresion>')';

ROUND: Esta función recibe como parámetro un valor numérico.

Estructura:

‘round ‘(‘ <Valor>‘)’;

LENGTH: Esta función recibe como parámetro un vector, una lista o una cadena y devuelve el tamaño deeste.

Estructura:

‘length ‘(‘ <Valor>‘)’;

TYPE_OF: Esta función retorna una cadena con el nombre del tipo de dato evaluado.

Estructura:

‘typeof ‘(‘ <Valor>‘)’;

TO_STRING: Esta función permite convertir un valor de tipo numérico o booleano en texto.

Estructura:

‘toString ‘(‘ <Valor>‘)’;

TO_CHAR_ARRAY: Esta función permite convertir una cadena en un vector de caracteres

Estructura:

‘toCharArray ‘(‘ <Valor>‘)’;

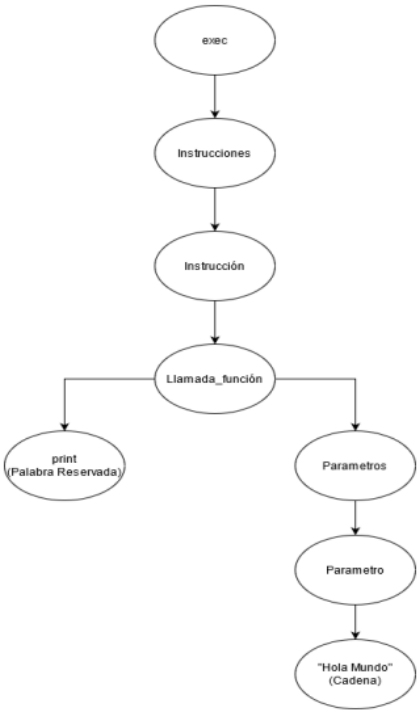
RUN: Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia RUN para indicar que método o función es la que iniciará con la lógica del programa.

Estructura:

‘run ‘(’ ’)’ ‘;’

‘run’ ‘(’ <LISTAVALORES> ’)’ ‘;’

LISTAVALORES-> LISTAVALORES ‘,’ EXPRESION | EXPRESION



a) Consola de Salida: En esta área se mostrarán los resultados, mensajes y todo lo que sea indicado dentro del lenguaje.

I) REPORTES

a) **Reporte de Errores**: Se mostrarán todos los errores encontrados al realizar el análisis léxico, sintáctico y semántico.

#	Tipo de Error	Descripción	Línea	Columna
1	Léxico	El carácter “$” no pertenece al lenguaje.	7	32
2	Sintáctico	Encontrado Identificador “Ejemplo”, se esperaba Palabra Reservada “Valor”	150	12

b) **Generar Árbol AST** (Árbol de Análisis Sintáctico): se debe generar una imagen del árbol de análisis sintáctico que se genera al realizar los análisis.

c) **Reporte de Tabla de Símbolos**: Se mostrarán todas las variables, métodos y funciones que han sido declarados dentro del flujo del programa.

II) SALIR

Detiene la ejecución del software.

Identificador	Tipo	Tipo	Entorno	Línea	Columna
Factor1	Variable	Entero	Función multiplicar	15	4
Factor2	Variable	Decimal	Función multiplicar	16	7
Resultado	Variable	Decimal	Función multiplicar	17	7
MostrarMensaje	Método	Void	-	50	6