GRAMÁTICA PROYECTO FLP

INTEGRANTES

VICTOR MANUEL MARÍN DUQUE
2015556071

DIEGO DANILO DELGADO
201556272

ANDRÉS CAMILO PULGARÍN
201556017

DOCENTE

CARLOS ANDRÉS DELGADO

UNIVERSIDAD DEL VALLE SEDE TULUÁ
FACULTAD DE INGENIERÍA Y COMPUTACIÓN
FUNDAMENTOS DE LENGUAJES DE PROGRAMACIÓN
TULUÁ - VALLE
MARZO 2020

ESPECIFICACIÓN GRAMATICAL DEL LENGUAJE EN FORMA DE BACKUS-NAUR (BNF)

La siguiente sintaxis se basa en el lenguaje de programación JavaScript; sin embargo, algunas características se modifican debido a los problemas por la izquierda (*LL*) de la librería *SLLGEN*.

<program> ::= <expresion>
un-programa (expresion)

<expresion> ::= <numero>
numero-exp (numero)

::= <identificador>
identificador-exp (identificador)

::= “<identificador>”
string-exp (identificador)

::= <flotante>
flotante-exp (flotante)

::= <”0x” 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}*>
hexadecimal-exp (hexadecimal)

::= <”0o” 0|1|2|3|4|5|6|7 {0|1|2|3|4|5|6|7}*>
octal-exp (octal)

::= var ({<type-exp> <identificador> = <expresion>}*(,))
definicion-exp (listatipos identificadores valores)

::= if (<expresion>) <expresion> else <expresion>
condicional-exp (condicion sentencia-verdad sentencia-falsa)

::= length (<expresion>)
longitud-exp (cadena)

::= concat (<expresion> <expresion>)
concatenacion-exp (cadena1 cadena2)

::= function <identificador> ({<type-exp> <identificador>}*(,))
<expresion>
procedimiento-exp (listatipos nombre-funcion parametros cuerpo)

::= call <identificador> ({<expresion>}*(,))

invocacion-proc-exp (nombre-funcion argumentos)

::= function-rec <identificador> ({<type-exp> <identificador>}*(,))
<expresion>
procedimiento-rec-exp (listatipos nombre-funcion parametros cuerpo)

::= call-rec <identificador> ({<expresion>}*(,))
invocacion-proc-rec-exp (nombre-funcion argumentos)

::= for (<expresion>; <expresion>; <expresion>) <expresion>
iteracion-exp (inicial-exp, condicion-for, incrementador, cuerpo)

::= <expresion> <primitiva-aritmetica> <expresion>
primitiva-aritmetica-exp (componente1 operando componente2)

::= <expresion> <primitiva-booleana> <expresion>
primitiva-booleana-exp (componente1 operando componente2)

::= true
verdad-exp ()

::= false
falso-exp ()

::= val <expresion> = <expresion>
asignacion-exp (identificador nuevo-valor)

::= { { <expresion>} }* (;)
secuenciacion-exp (lista-exp)

::= struct <identificador> {{<type-exp> <identificador> = <expresion>}* (;)}
estructura-exp (nombreestructura lista-exp)

::= access <identificador> [<numero>]
acceso-estructura (nombreestructura posicion)

<primitiva> ::= + | - | * | % | / |< | > | <= | >= | == | != | && | ||

<primitiva2> ::= ++ | --

<type-exp> ::= int
int-type ()

::= float
float-type ()

::= hex
hexadecimal-type ()

::= oct
octal-type ()

::= string
string-type ()

::= bool
bool-type ()

::= proc ({ <type> }* (*) -> <type>)
procedimiento-type ()