Lenguaje de programación M4

➔ ***Integrantes:***

Miguel Ángel Jerónimo Mejía	00004412
Kattya Leonor Martínez Rivas	00023512
Miguel Antonio Montes Hernández	00020612
Luis Fernando Orellana Morales	00014312

➔ ***Información básica.***

◆ ***Diseño del lenguaje.***

M4 fue diseñado bajo la influencia de los lenguajes de programación que han innovado conforme el paso del tiempo el estilo de muchos desarrolladores a nivel mundial. Se basa primeramente en la programación estructurada así como en una sintaxis ordenada que facilita la legibilidad y el discernimiento al momento de establecer el flujo de los datos.

◆ ***Posibles aplicaciones.***

Se puede utilizar para desarrollar aplicaciones cuya principal característica sea la eficiencia en cuanto a uso de recursos, así como fáciles de entender, reducir los costos de mantenimiento e incrementar el rendimiento de los equipos de desarrollo.

◆ ***Descripción de la sintaxis.***

La sintaxis de M4 es bastante sencilla.

```
1  main{
2      console.write("Hola Mundo");
3  }
```

El programa principal debe comenzar con `main {` y finalizar con `}` cabe mencionar que:

- Todas las líneas de código deben finalizar con ;
- El compilador no es case sensitive, por lo que no es necesario tomar en consideración la forma en cómo se escriben las palabras (de preferencia en minúsculas) .

◆ ***Palabras reservadas.***

var.int	default	func.char	main	sqrt	cos
var.float	break	func.str	var.file	substring	tan
var.char	while	func.bool	file.fopen	pow	arcsin
var.str	do	func.void	file.fscanf	length	arccos

var.bool	for	arr.int	file.fgets	concat	arctan
if	console.write	arr.float	file.fclose	replace	log
else	console.read	arr.char	split	equal	true
switch	func.int	arr.str	trim	sin	false
case	func.float	arr.bool	round		

➔ *Expresión regular para los identificadores*

La variable debe empezar con un carácter y luego seguir de n cantidad de números o letras, donde n, que corresponde al tamaño máximo definido que tendrá.

Para utilizar una variable basta con colocar el nombre de la misma.

La declaración de variables será de la forma *<Tipo de Dato> <Nombre de Variable>*

La asignación de valores para las variables se hará con la forma

<variable> := <valor>

Donde los tipos de datos son los siguientes:

Entero, Flotante, Carácter, Cadena, Booleano, FILE

```
var.int variableEntero := 12;
var.float variableFloat := 12.5;
var.char variableChar := "a";
var.str variableString := "hola mundo";
var.bool variableBool := true;
```

Expresión Regular: (“var.int”+ “var.float”+ “var.char”+ “var.str”+ “var.bool”)

Expresión Regular: letra(letra+dígito+_)*

➔ *Expresiones regulares para números enteros y números reales*

Los números se declararán de la siguiente manera:

var.int variable_entero := 1;
var.float variable_flotante := 1.2;

Las expresiones regulares que se utilizaron:

Número entero	(número)+
Número flotante	(número)+.(número)+

➔ *Lista de operadores y caracteres especiales*

Operadores			
:=	Asignación	+=	Más igual
>	Mayor que	<	Menor que
>=	Mayor o igual	<=	Menor o igual
==	Comparación (Igual que)	!=	Diferente de
&&	Operador logico (AND)	||	Operador lógico (OR)

Caracteres especiales					
&	{	}	(	)	[
]	:	.	,	+	-
*	/				

➔ ***Forma de construcción de comentarios en el lenguaje***

Los comentarios en M4 se escriben comenzando con el carácter #, solamente posee comentarios de una línea, pero se pueden colocar muchos de forma consecutiva. Ejemplo:

```
1  #Esto es un comentario
2  #Este es otro comentario
3  #Se pueden colocar muchos comentarios de esta forma.
```

➔ ***Limitaciones***

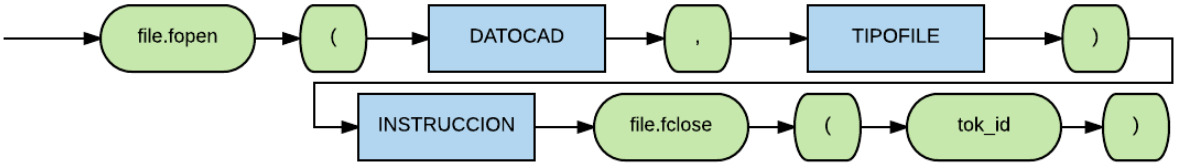
- El tamaño máximo de las cadenas es limitado por el compilador, por el momento. (200 caracteres).

➔ ***Diagramas de sintaxis - Notación BNF.***

- ::= se utilizará como definición (derecha produce izquierda).
- | se utilizará como alternativa entre diferente elementos.
- { } significa que el elemento será repetido 0 o más veces.
- [ ] es de opción, que puede utilizarse o no.
- ( ) es para agrupar los elementos que incluye.
- Los símbolos terminales serán representados entre los caracteres “ “
- Los símbolos no terminales serán representados entre los símbolos < >

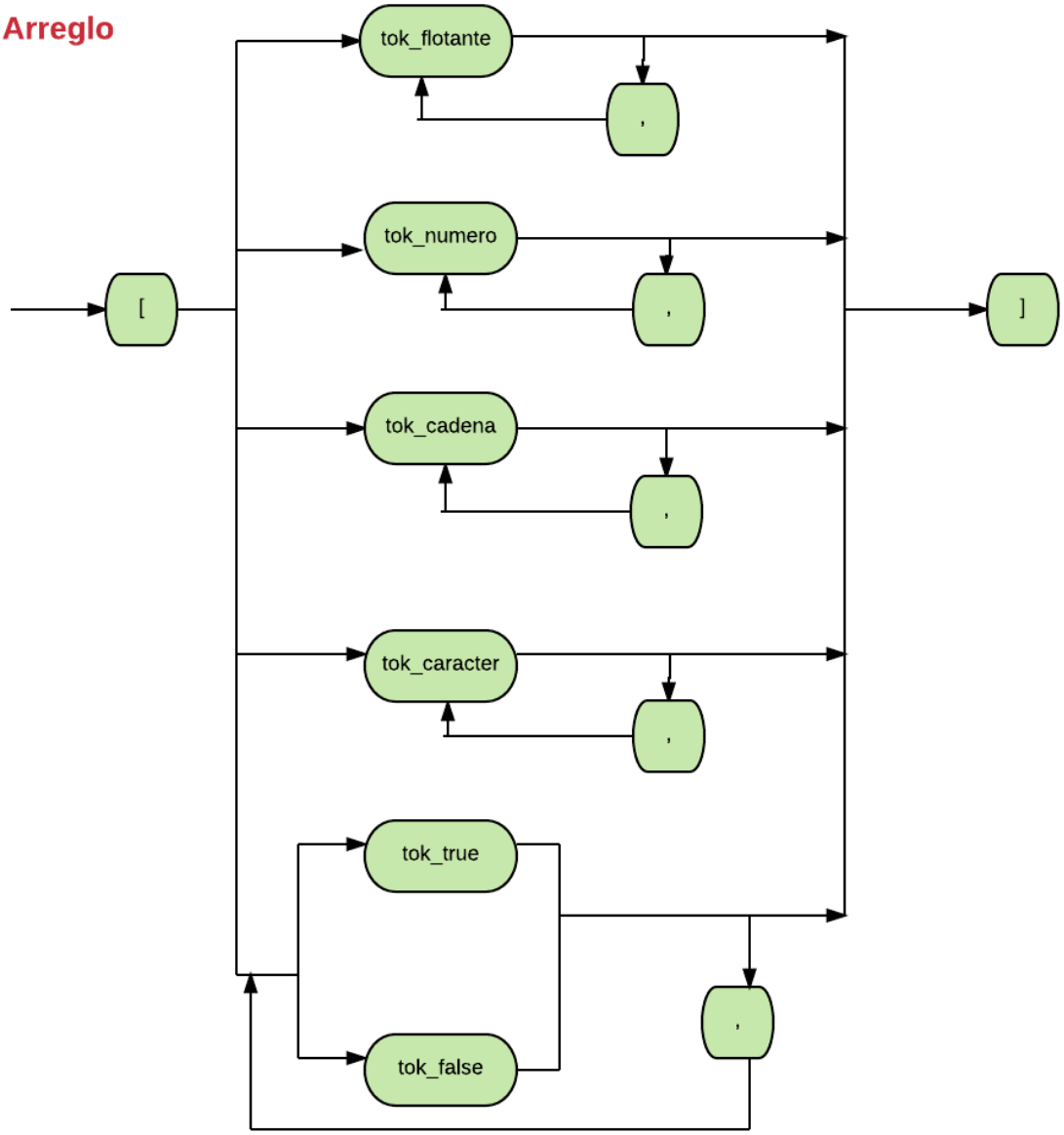
Abrir archivo

AbrirArchivo



<AbrirArchivo>::=“ file.open““(“<DatoCad> “ , “ < TipoFile > “) “ <Instrucción>”file.fclose” “(“ “tok_id” “)”

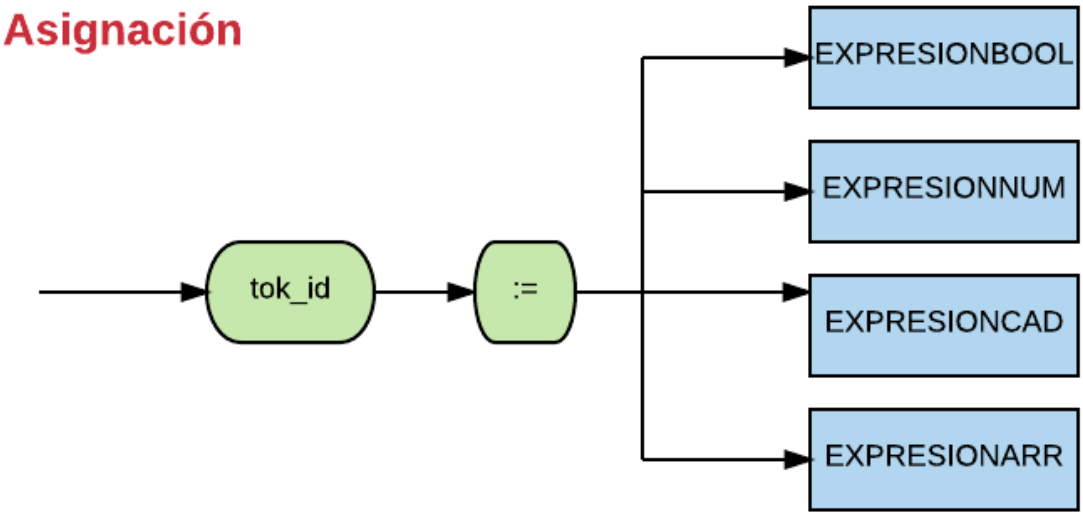
Arreglo



< Arreglo > ::= ” [” “ tok_flotante ” { (“, “ “ tok_flotante ”)} “] ” |
” [” “ tok_numero” { (“, “ “ tok_numero”)} “] ” |
” [” “ tok_cadena” { (“, “ “ tok_cadena”)} “] ” |
” [” “ tok_caracter” { (“, “ “ tok_caracter”)} “] ” |
” [” <Booleano> { (“, “ “ <Booleano> ”)} “] ”

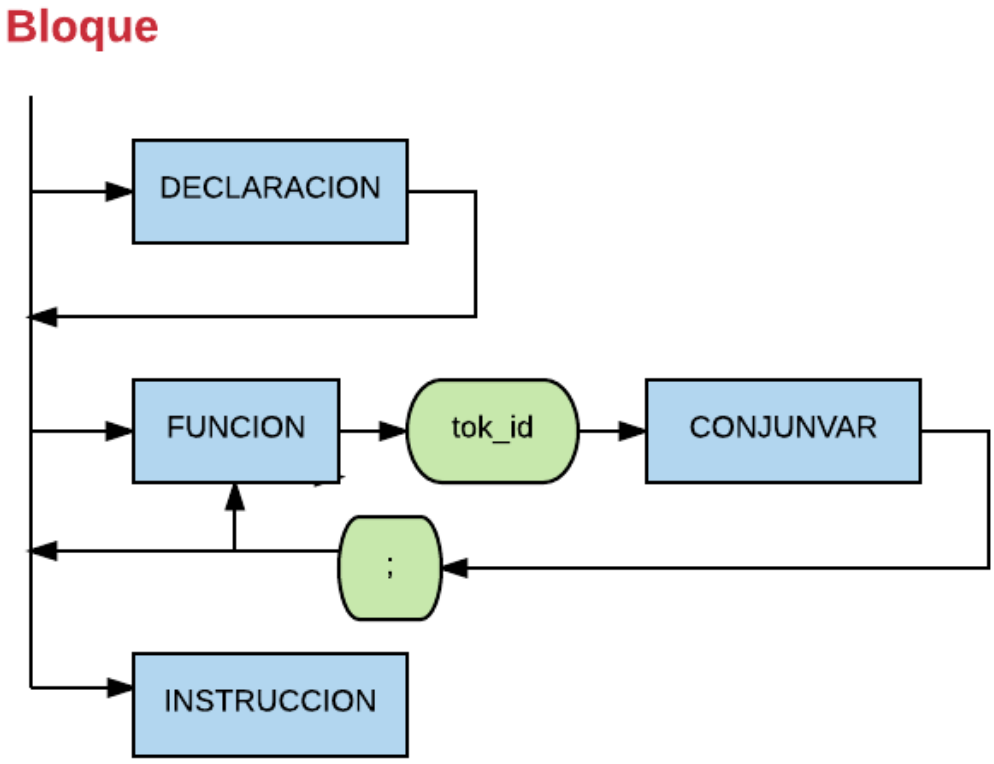
<Booleano> ::= “tok_true” | “tok_false”

Asignación



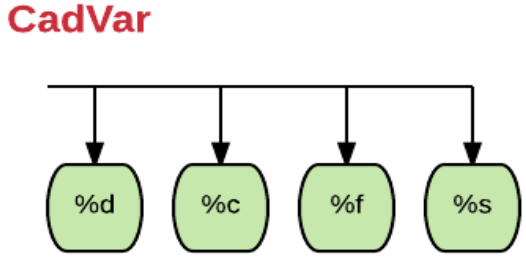
<Asignacion> ::= " tok_id " " := " <ExpresionGeneral>
<ExpresionGeneral> := < ExpresionBool> |
<ExpresionNum> |
<ExpresionCad> |
<ExpresionArr>

Bloque



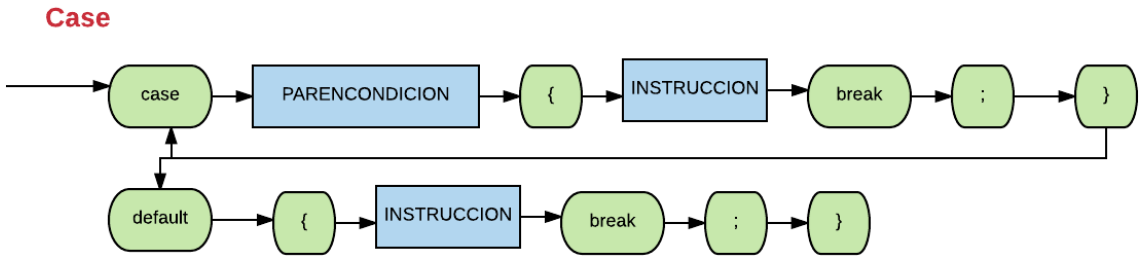
< Bloque > ::= [< Declaracion >] [< DeclaracionFuncion >] < Instruccion >
< DeclaracionFuncion>::=< Funcion > " tok_id "< ConjunVar >" ; " {< DeclaracionFuncion>}

CadVar



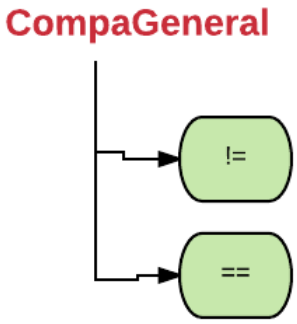
<CadVar> ::= “ %d ” | “ %c ” | “ %f ” | “ %s”

Case



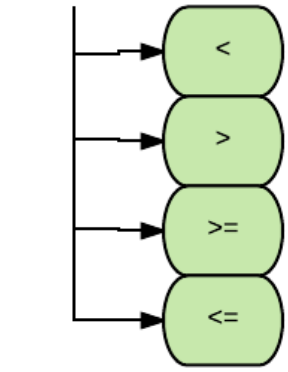
<Case> ::= <DefinicionCase>{ <DefinicionCase>} “default” “ { “ <Instruccion> “break” “;” “}”
<DefinicionCase> ::= “ case” <ParenCondicion > “ { “ < Instruccion > “break” “ ; ” “ } ”

CompaGeneral



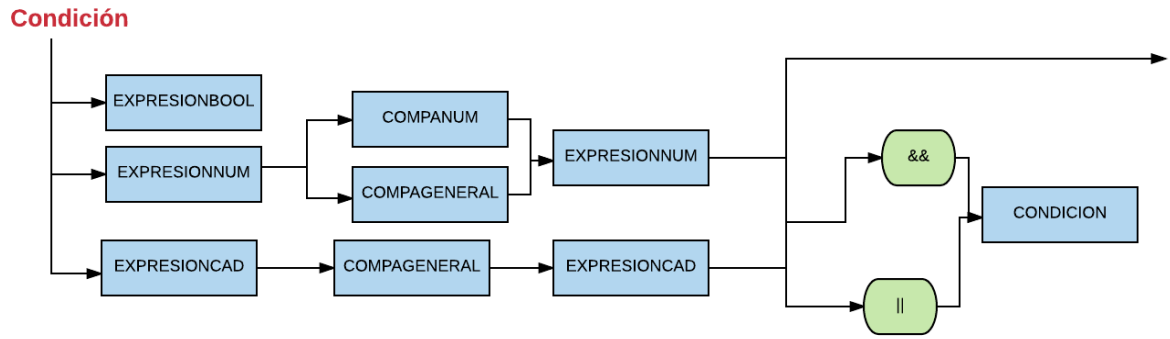
<CompaGeneral> ::= “ != ” | “ == ”

CompaNum



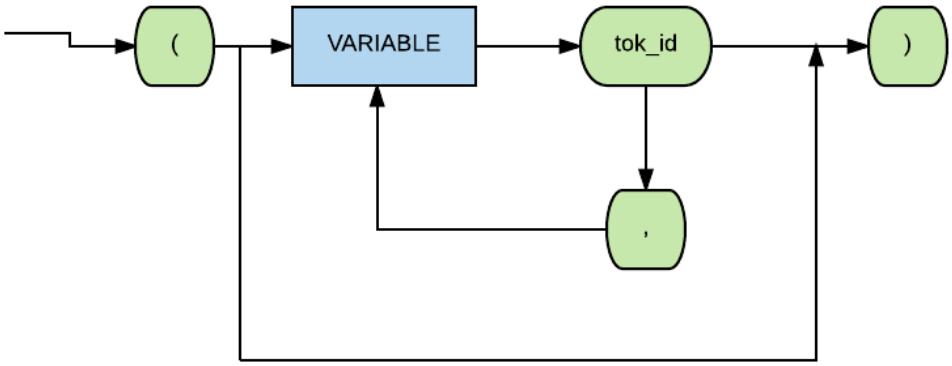
<CompaNum> ::= “ < “ | “ > “ | “ >= “ | “ <=”

Condición



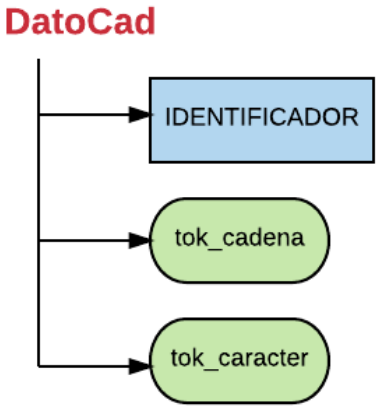
< Condición > ::= <CondicionGeneral > { (< OperadorLogico > <CondicionGeneral>) }
< OperadorLogico > ::= " && " | " || "
< CondicionGeneral > ::= < ExpresionBool > |
< ExpresionNum > <Comparación> <ExpresionNum> |
< ExpresionCad > <CompaGeneral > <ExpresionCad >
<Comparación > ::= <CompaGeneral > | <CompaNum >

ConjunVar



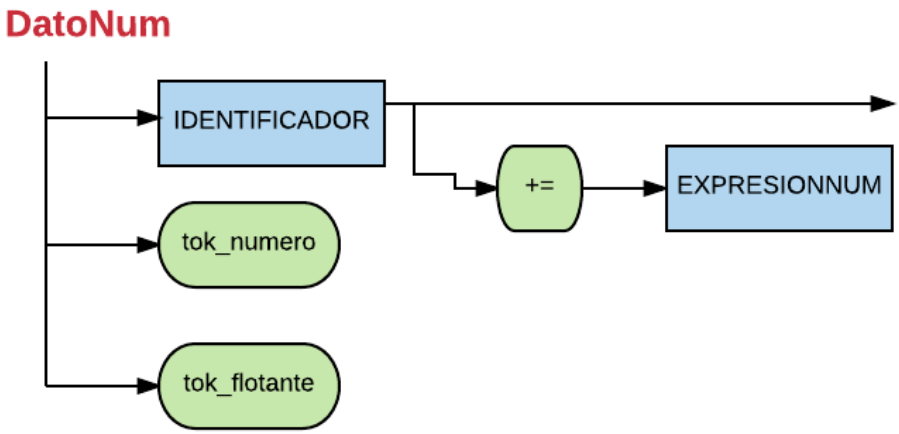
< ConjunVar > ::= " (" [<Variable> " tok_id" { (", " < Variable >)}] ") "

DatoCad



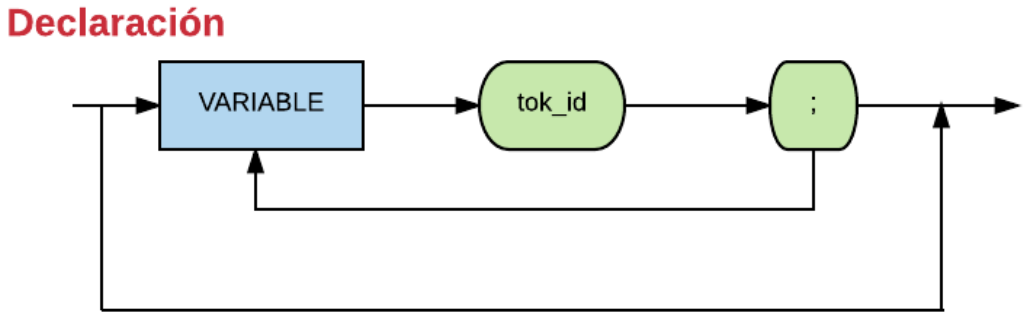
< DatoCad > ::= < Identificador > | " tok_cadena" | "tok_caracter"

DatoNum



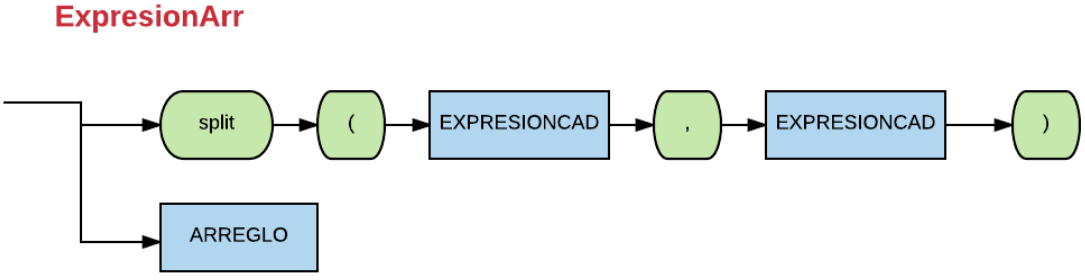
< Dato Num > ::= < Identificador > [(" += " <ExpresionNum>)] |
" tok_numero " |
" tok_flotante "

Declaración



< Declaración > ::= [(< Variable > " tok_id " ";" { (< Variable> " tok_id " " ; ")})]

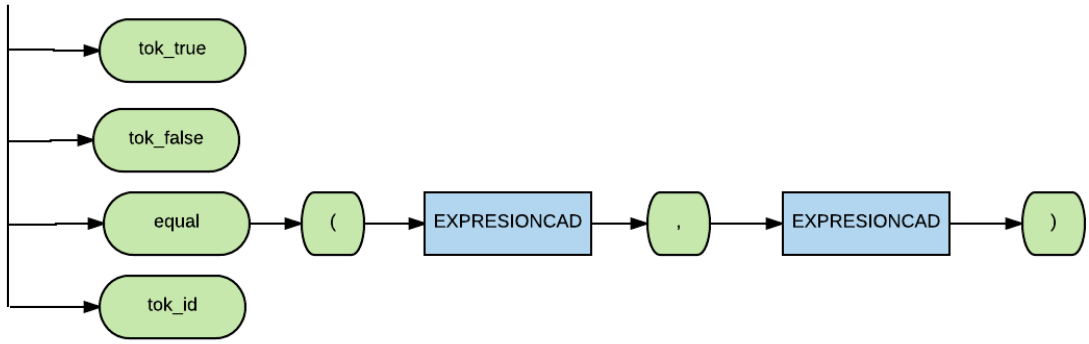
Expresión arreglo



< ExpresionArr > ::= " split " " (" <ExpresionCad > ", " < ExpresionCad> ") " |
<Arreglo >

Expresión bool

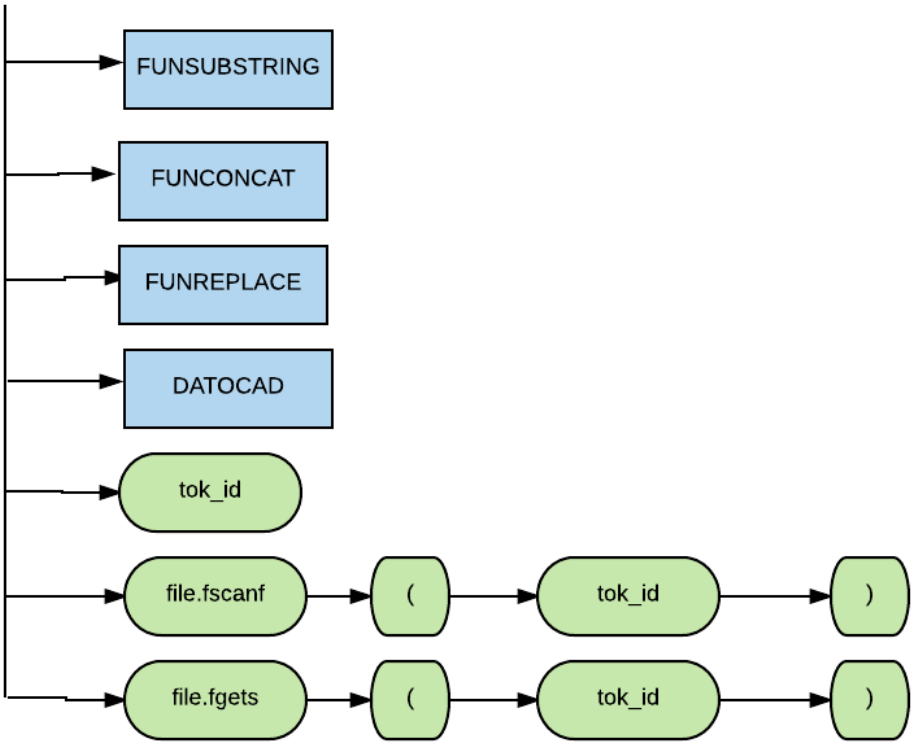
ExpresionBool



< ExpresionBool > ::= *“ tok_true ” |*
“ tok_false “ |
“ equal” “ (“ < ExpresionCad > “, “ < ExpresionCad > “) “ |
“ tok_id “

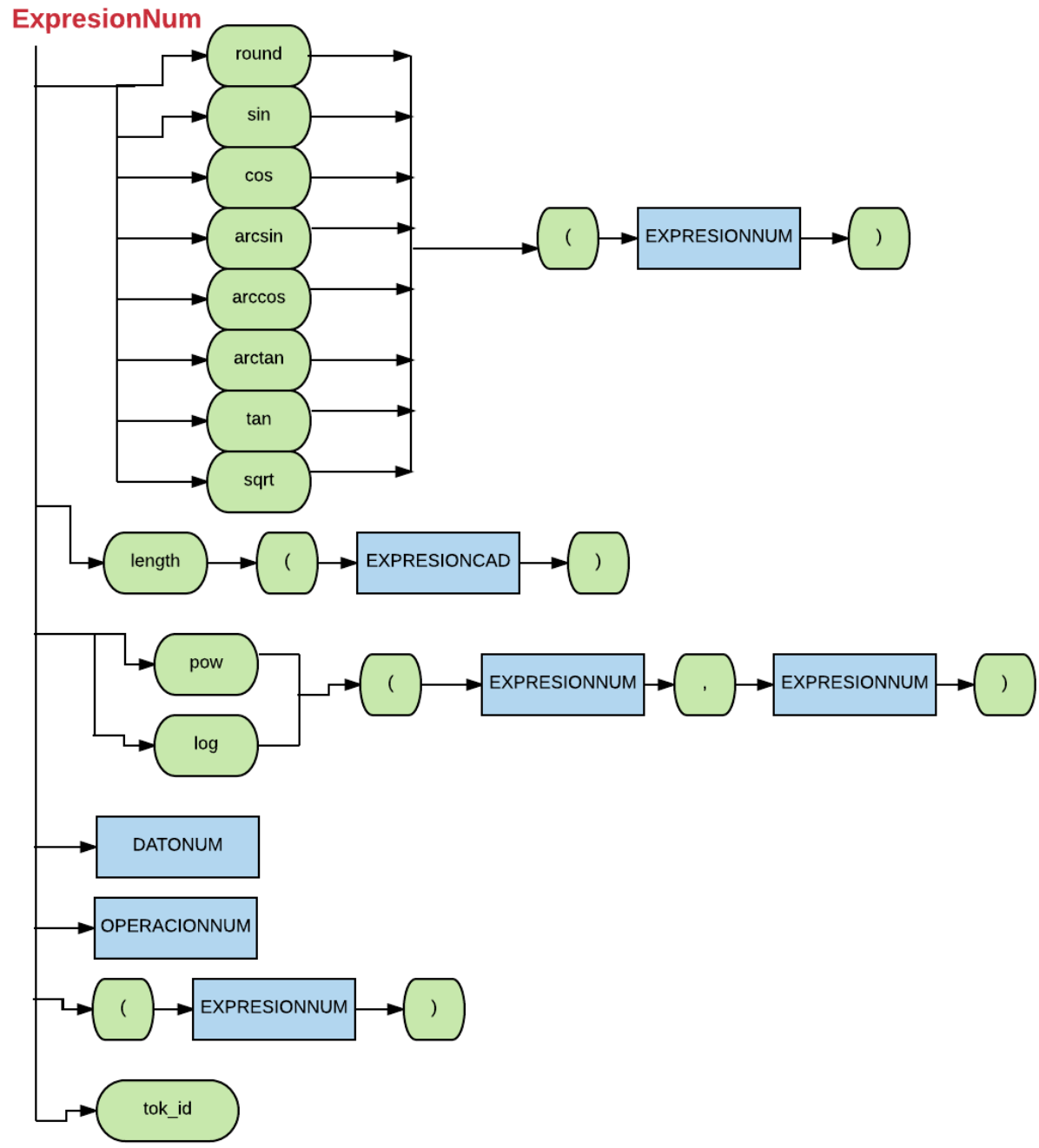
Expresión Cadena

ExpresionCad



< ExpresionCad > :: = *< FunSubString > |*
< FunConcat > |
< FunReplace > |
< DatoCad > |
“ tok_ id “ |
“ file.fscanf “ “(“ “tok_id “ “)” |
“ file.fgests “ “(“ “ tok_id “)”

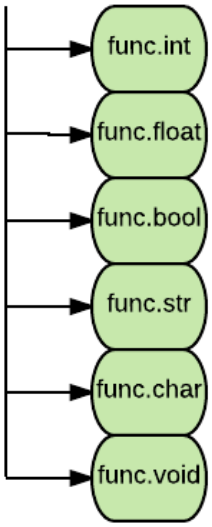
Expresión Número



< ExpresionNum > ::= *<OperMatematico> “ (“ <ExpresionNum> “) ” |*
“ length “ “ (“ < ExpresionCad > “) “ |
“ pow “ “(“ < ExpresionNum > “, “ <ExpresionNum > “)” |
“ log “ “(“ < ExpresionNum > “, “ <ExpresionNum > “)” |
<DatoNum > |
<OperacionNum>|
“(“ <ExpresionNum > “) “ |
“ tok_id “

Función

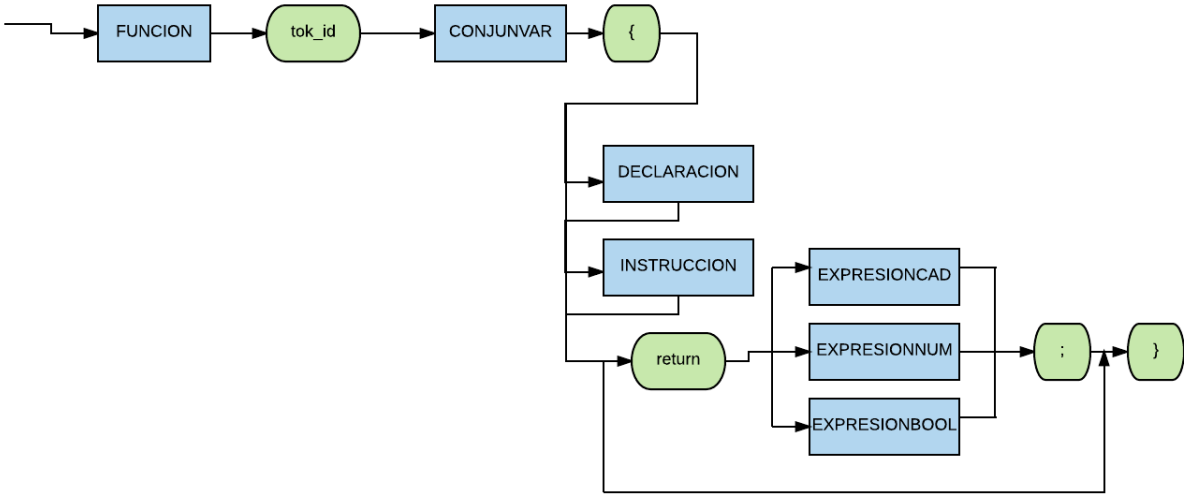
Funcion



< Función > ::= “ func.int “ | “func.float “ | “func.bool” | “func.str” | “func.char” | “func.void”

Función instrucción

Función Instrucción



< FuncionInstruccion > ::= < Funcion > “ tok_id” < ConjunVar > “ (“ [< Declaracion >]
< Instruccion > [(“ return “ < ExpresionInstruccion> “ ; ”)] “ } ”

<ExpresionInstruccion> ::= < ExpresionCad > |
< ExpresionNum > |
< ExpresionBool >

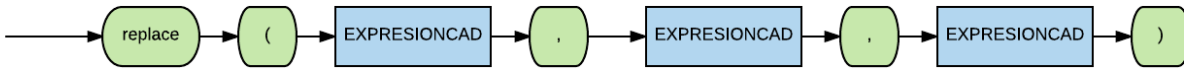
Función concatenar

Funconcat

< FunConcat > ::= “ concat “ “ (“ < ExpresionCad > “ , “ <ExpresionCad > “)”

Función reemplazar

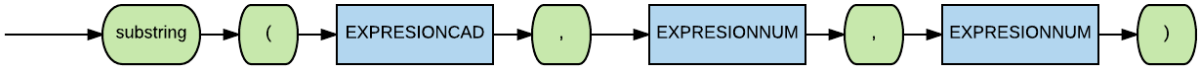
Funreplace



< FunReplace > ::= “ replace “ “(“ < ExpresionCad > “ , “ < ExpresionCad > “ , “ < ExpresionCad > “) “

Función substring

Funsubstring



< FunSubString > ::= “substring” “ (“ <ExpresionCad> “,” <ExpresionNum > “, “ <ExpresionNum > “) “

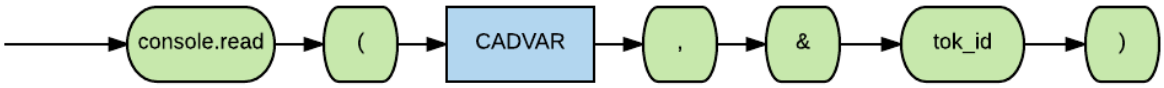
Identificador

Identificador

<Identificador > ::= “ tok_id “ [(“[“ <ExpresionNum >)]

Instrucción console read

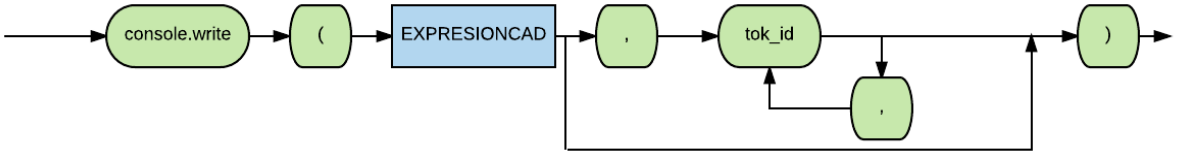
InsConsoleRead



< InsConsoleRead > ::= “console.read” “ (“ < CadVar > “, “ “ &” “tok_id “ “) “

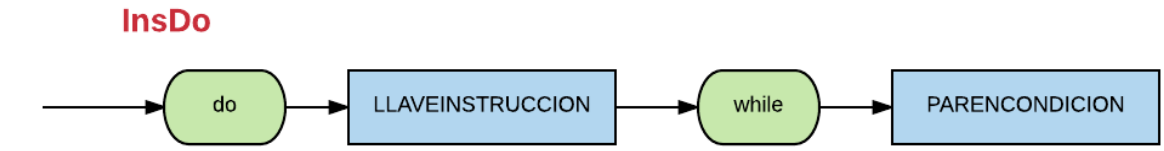
Instrucción console write

InsConsoleWrite



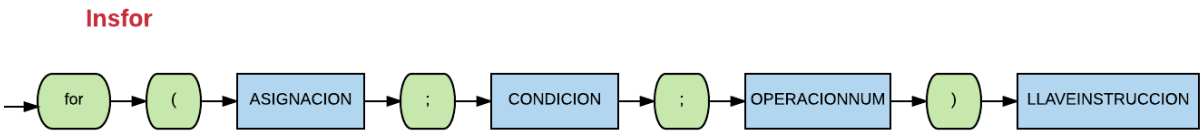
<InsConsoleWrite>::=“console.write” “(“< ExpresionCad > [“,” “ tok_id “ {(“,“ “ tok_id “)} “)“]

Instrucción do



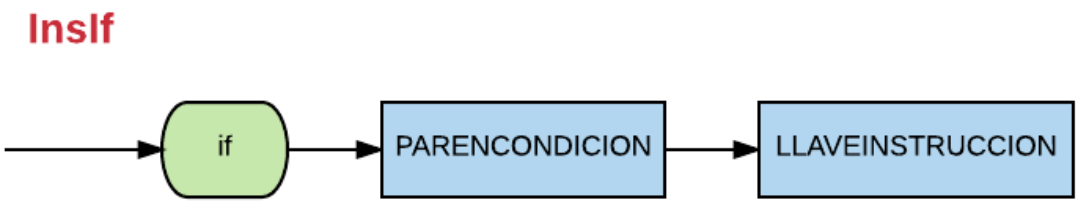
< InsDo > ::= “do “ < LlaveInstruccion > “ while “ < ParenCondicion >

Instrucción for



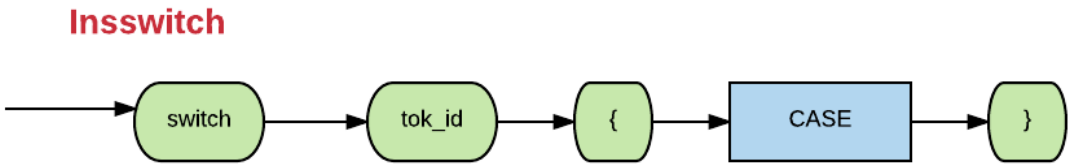
< InsFor > ::= “for “ “ (“ <Asignacion> “; “ <Condicion> “; “ <Operacionnum> “) <LlaveInstruccion>

Instrucción if



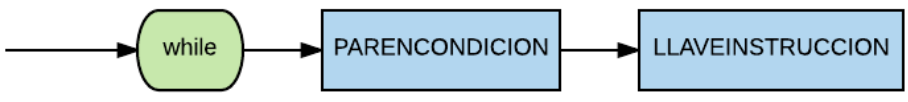
< InsIf > ::= “ if “ <ParenCondicion> <LlaveInstruccion >

Instrucción switch



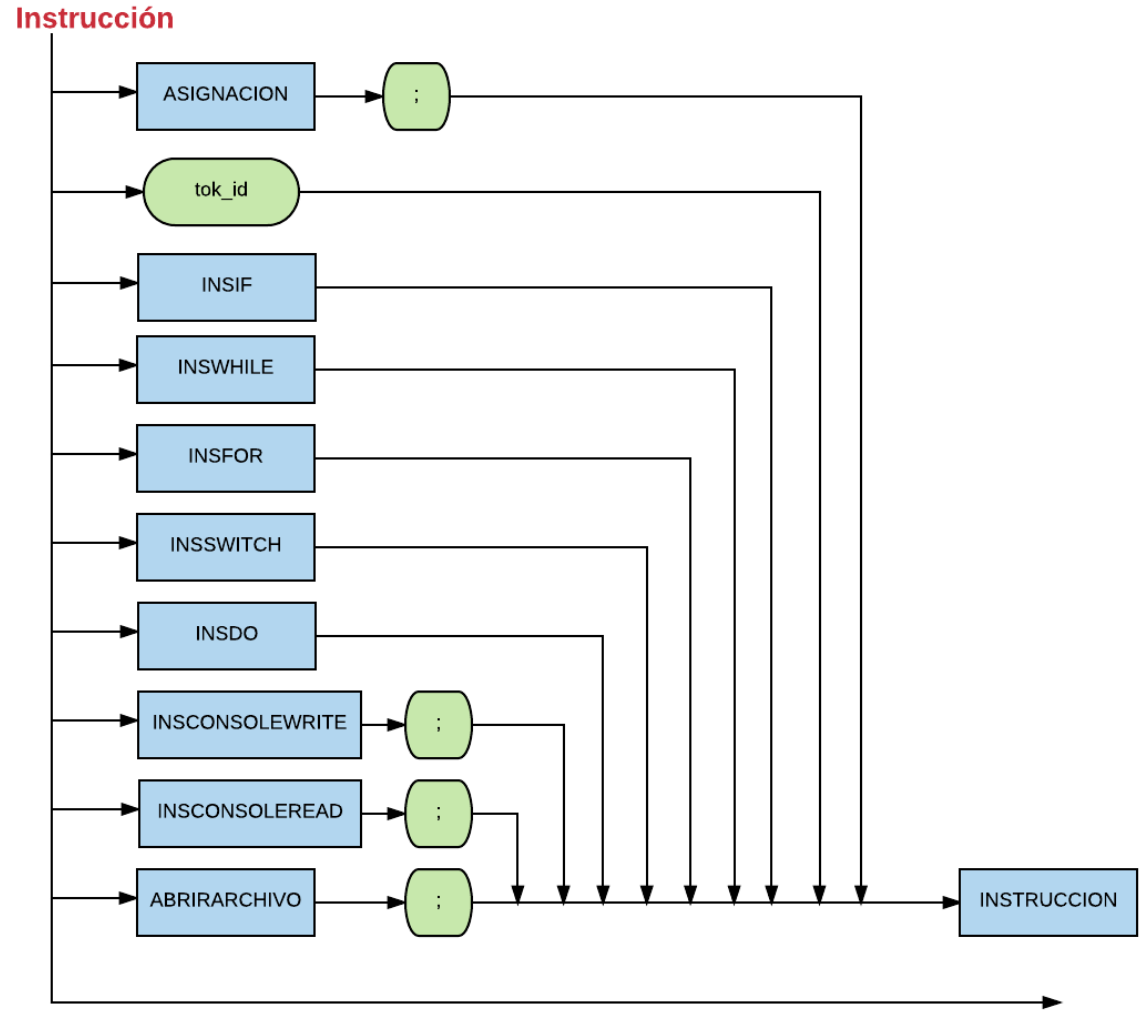
<InsSwitch > ::= “switch “ “tok_id” “ {“ <Case > “}”

Instrucción while



< InsWhile > ::= “while” <ParenCondicion> <LlaveInstruccion>

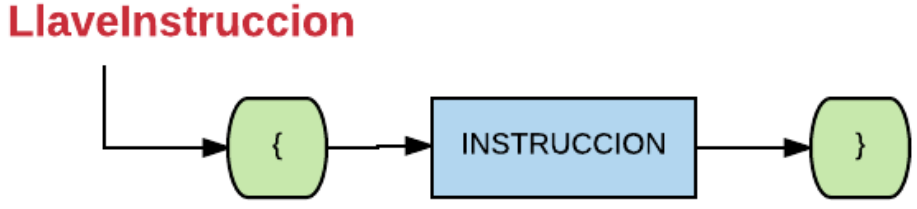
Instrucción



< Instruccion > ::= { < InstruccionGnal >}

< InstruccionGnal > ::= *<Asignacion> " ;" |*
"tok_id" |
< InsIf > |
< InsFor > |
<InsSwitch > |
<InsDo > |
<InsConsoleWrite> ";" |
<InsConsoleRead> ";" |
<AbrirArchivo> ";" |

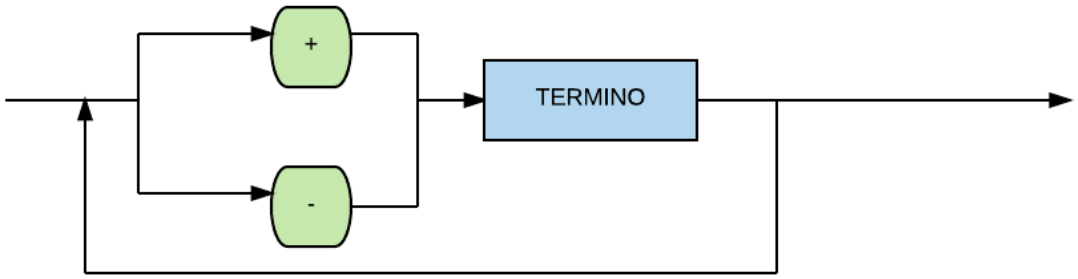
LLave instrucción



< LlaveInstruccion > ::= " { " < Instruccion > " } "

Operación número

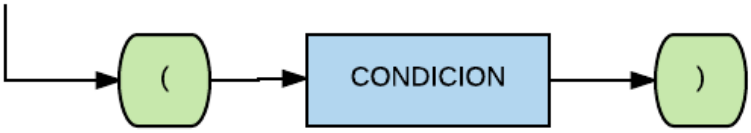
OperacionNum



< OperacionNum > ::= < Operadores > < Termino> { (<Operadores > <Termino >) }
< Operadores > ::= “ + ” | “ - ”

ParenCondición

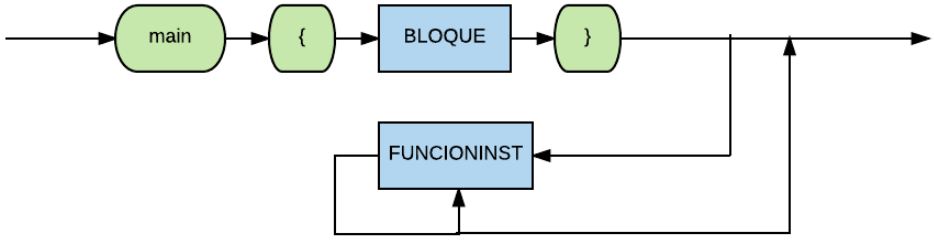
ParenCondicion



< ParenCondicion > ::= “ (“ < Condicion > “) “

Programa

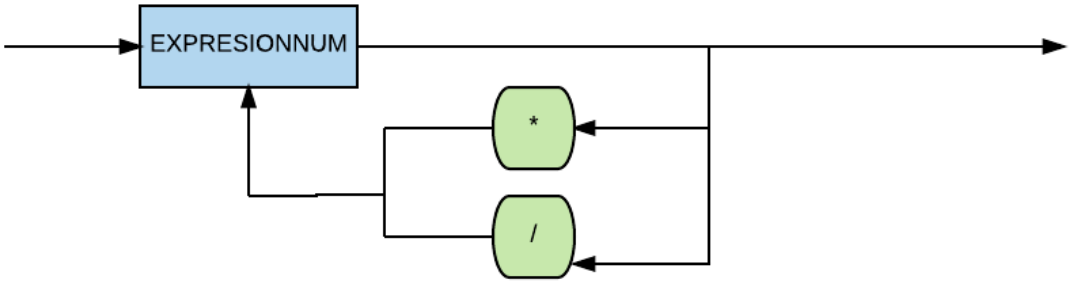
Programa



< Programa > ::= “main” “ {“ < Bloque > “ } { < FuncionInst > }

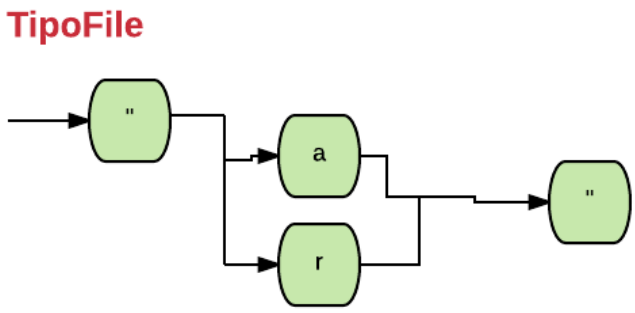
Término

Termino



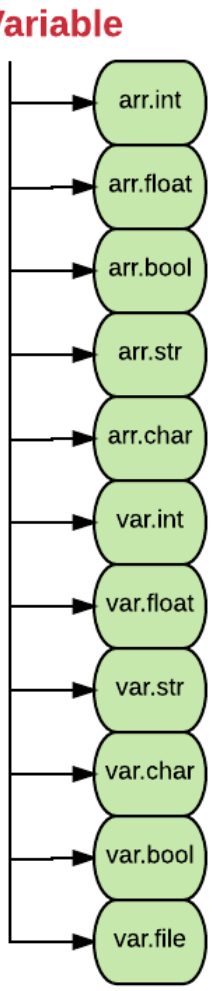
< Termino > ::= <ExpresionNum > { (< MultiDiv > < ExpresionNum >) }
*< MultiDiv > ::= “ * ” | “ / ”*

Tipo archivo



< TipoFile > ::= *“ “ “ a “ “ “ |*
“ “ “ r “ “ “

Variable



< Variable > ::= *“ arr.int” | “ arr.float ” | “ arr.bool ” | “ arr.str ” | “ arr.char ” |*
“ var.int” | “ var.float “ | “var.str” | “var.char “ | “var.bool” | “var.file”

➔ *Descripción de gramática- Notación matemática y aplicación de regla #1 y # 2 de comprobación de gramática LL (1)*

PROGRAMA

PROGRAMA → main { BLOQUE } FUNCINST

PRIM (PROGRAMA) = { main }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

Función Instrucción → FUNCINST

FUNCINST ❼ ∑ |
FUNCION tok_id CONJUNVAR { DECLARACION INSTRUCCIÓN RETORNO }

PRIM (FUNCINST) = PRIM(FUNCION)
PRIM (FUNCINST) = { func.int, func.char, func.str, func.bool, func.float }

Regla #1 : *Ø ∩ PRIM(FUNCION) = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(FUNCINST) ∩ SIG(FUNCINST) = ¿?*

={ func.int, func.char, func.str, func.bool, func.float } ∩ Ø = Ø

Cumple con la regla #2 ¡!

DECLARACION

DECLARACION ❼ ∑ |
VARIABLE tok_id ;

PRIM (DECLARACION) = PRIM(VARIABLE)
PRIM(DECLARACION) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1: *Ø ∩ PRIM(DECLARACION) = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(DECLARACION) ∩ SIG(DECLARACION) = ¿?*

SIG(DECLARACION) = PRIM(INSTRUCCIÓN) U PRIM(RETORNO) U PRIM(FUNCIONDECLA)

= { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} U { return } U { func.int, func.char, func.str, func.bool, func.float }

PRIM(DECLARACION) ∩ SIG(DECLARACION) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file } ∩ { tok_id, if, while, switch, do, console.write, console.read, file.fopen, return, func.int, func.char, func.str, func.bool, func.float } = Ø . Cumple con la regla #2.

RETORNO

RETORNO ❼ ∑ |
return EXPRESION ;

PRIM(RETORNO) = { return }

Regla #1 : *Ø ∩ { return } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(RETORNO) ∩ SIG(RETORNO) = ¿? = { return }∩ { } } = Ø. Cumple con la regla #2.*

VARIABLE

VARIABLE ❼ arr.int | arr.bool | arr.char | arr.str | arr.float |
var.int | var.bool | var.char | var.str | var.float | var.file

PRIM (VARIABLE) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1 : *Cumple con la regla #1 y no produce la palabra vacía*

FUNCION

FUNCION → func.int | func.bool | func.char | func.str | func.float

PRIM(FUNCION) = { func.int, func.char, func.str, func.bool, func.float }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía.*

CONJUNVAR

CONJUNVAR → (VARIABLE tok_id MASCONJUNVAR)

PRIM(CONJUNVAR) = { (}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía.*

MASCONJUNVAR

MASCONJUNVAR ❼ ∑ |
, VARIABLE tok_id

PRIM (MASCONJUNVAR) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCONJUNVAR) ∩ SIG(MASCONJUNVAR) = ¿? = { , } ∩ {) }= Ø . Cumple la regla #2.*

BLOQUE

BLOQUE → DECLARACION FUNCIONDECLA INSTRUCCIÓN

PRIM(BLOQUE) = PRIM(DECLARACION) U PRIM(FUNCIONDECLA) U PRIM(INSTRUCCIÓN)

PRIM(BLOQUE) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1 : *Cumple con la regla #1 y no produce la palabra vacía*

FUNCIONDECLA

FUNCIONDECLA → e | FUNCION tok_id CONJUNVAR ; FUNCIONDECLA

PRIM(FUNCIONDECLA) = PRIM(FUNCION)

PRIM(FUNCIONDECLA) = { func.int, func.char, func.str, func.bool, func.float }

Regla #1: *Ø ∩ { func.int, func.char, func.str, func.bool, func.float } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(FUNCIONDECLA) ∩ SIG(FUNCIONDECLA) = ¿?*

= { func.int, func.char, func.str, func.bool, func.float } ∩ PRIM(INSTRUCCIÓN) = Ø .

= { func.int, func.char, func.str, func.bool, func.float } ∩ { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} = Ø . Cumple la regla #2.

EXPRESION

EXPRESION → EXPRESIONCAD | EXPRESIONNUM | EXPRESIONBOOL

PRIM (EXPRESION) = PRIM(EXPRESIONCAD) U PRIM(EXPRESIONNUM) U PRIM(EXPRESIONBOOL)

= { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets } U { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (} U { true, false, equal,tok_id }
= { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, entero, flotante, +, - , (, true, false, equal, tok_id }

Regla # 1: *Ø) PRIM(EXPRESIONCAD)) PRIM(EXPRESIONNUM) PRIM(EXPRESIONBOOL) = { tok_id }*
Expresion no es LL1, Sin embargo se puede solucionar aplicando reglas semánticas. Se tomará el tok_id de EXPRESIONCAD si el identificador en la tabla de símbolos es de tipo func.str, func.char, var.char o var.str. Y el tok_id de EXPRESIONNUM se tomará si en la tabla de símbolos es de tipo func.int, func.float, var.int o var.float. Si es var.bool o func.bool se tomará EXPRESIONBOOL

EXPRESIONCAD

EXPRESIONCAD ❼

FUNSUBSTRING |
FUNCONCAT |
FUNREPLACE |
DATOCAD |
tok_ id | *función de tipo Cadena o carácter*
file.scanf (tok_id)
file.fgets (tok_id)

PRIM (EXPRESIONCAD) = PRIM(FUNCSUBSTRING) U PRIM(FUNCONCAT) U PRIM(FUNREPLACE) U PRIM(DATOCAD) U { tok_id, file.scanf, file.fgets }

PRIM (EXPRESIONCAD) = { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *{ substring } ∩ { concat } ∩ { replace } ∩ { tok_id, cadena, carácter } ∩ { tok_id, file.scanf, file.fgets } = { tok_id }*

Esta producción no es LL1, sin embargo puede corregirse aplicando reglas semánticas, se tomará el tok_id de ExpresionCad si en la tabla de símbolos es una func.str o func.char, en cambio si en la tabla de símbolos es un var.str o un var.char se tomará el camino de DATOCAD.

FUNSUBSTRING

FUNSUBSTRING → substring (EXPRESIONCAD ,EXPRESIONNUM , EXPRESIONNUM)

PRIM(FUNSUBSTRING) = { substring }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

FUNCONCAT

FUNCONCAT → concat (EXPRESIONCAD , EXPRESIONCAD)

PRIM(FUNCONCAT) = { concat }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

FUNREPLACE

FUNREPLACE → replace (EXPRESIONCAD , EXPRESIONCAD , EXPRESIONCAD)

PRIM(FUNREPLACE) = { replace }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

DATOCAD

DATOCAD ❼ IDENTIFICADOR |
Cadena |
Carácter

PRIM(DATOCAD) = PRIM(IDENTIFICADOR) U { cadena, caracter }
PRIM(DATOCAD) = { tok_id, cadena, carácter }

Regla #1: *PRIM(IDENTIFICADOR) ∩ { cadena, caracter} = Ø , Cumple la regla 1 y no produce la palabra vacía.*

IDENTIFICADOR

IDENTIFICADOR → tok_id IDENARR

PRIM(IDENTIFICADOR) = { tok_id }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

IDENARR

IDENARR ❼ ∑ **|**
[VALORARR]

PRIM(IDENARR) = { [}

Regla #1: *Ø ∩ { [} = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(IDENARR) ∩ SIG(IDENARR) = ¿? = { [} ∩ Ø = Ø . Cumple la regla #2.*

VALORARR

VALORARR ❼ **tok_id |** *Variable de tipo entero*
entero

PRIM (VALORARR) = { tok_id, entero }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

EXPRESIONNUM

EXPRESIONNUM ❼ **round (EXPRESIONNUM) |**
Sin (EXPRESIONNUM) |
Cos (EXPRESIONNUM) |
Arcsin (EXPRESIONNUM) |
Arccos (EXPRESIONNUM) |
Arctan (EXPRESIONNUM) |
Tan (EXPRESIONNUM) |
Sqrt (EXPRESIONNUM) |
Length (EXPRESIONCAD) |
Pow (EXPRESIONNUM , EXPRESIONNUM) |
Log (EXPRESIONNUM , EXPRESIONNUM) |
DATONUM |
OPERACIONNUM |
(EXPRESIONNUM) |
Tok_id *función entero o flotante*

PRIM (EXPRESIONNUM) = { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log } U PRIM(DATONUM) U PRIM(OPERACIONNUM) U { (, tok_id}

PRIM(EXPRESIONNUM) = { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (, }

Regla #1: { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log } *)* { tok_id, entero, flotante } *)* { +, - } *)((, tok_id } = { tok_ id }*

Esta producción no es LL1 sin embargo se puede solucionar con reglas semánticas, se tomará el camino del tok_id de EXPRESIONNUM si en la tabla de símbolos es una func.int o func.float, de lo contrato si en la tabla de símbolos es un var.int o un var.float se tomará el camino de DATONUM.

DATONUM

DATONUM ➐ IDENTIFICADOR SUMATERMINO| *Variable de tipo entero o flotante*
Entero |
Flotante

PRIM(DATONUM) = PRIM(IDENTIFICADOR) U { entero, flotante }
PRIM(DATONUM) = { tok_id, entero, flotante }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

OPERACIONNUM

OPERACIONNUM ➐ OPERATERMINO TERMINO MASOPERACIONNUM

PRIM(OPERACIONNUM) = PRIM (OPERATERMINO)
PRIM(OPERACIONNUM)= { +, - }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASOPERACIONNUM

MASOPERACIONNUM ➐ ∑ | OPERATERMINO TERMINO MASOPERACIONNUM

PRIM(MASOPERACIONNUM) = PRIM(OPERATERMINO) = { +, - }

Regla #1: *Ø ∩ { +, - }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASOPERACIONNUM) ∩ SIG(MASOPERACIONNUM) = ¿? = { +, - } ∩ Ø = Ø . Cumple la regla #2.*

OPERATERMINO

OPERATERMINO ➐ + | -
PRIM (OPERATERMINO) = { + , - }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

SUMATERMINO

SUMATERMINO ➐ ∑ |
+ = EXPRESIONNUM
PRIM (SUMATERMINO) = { += }

Regla #1: *Ø ∩ { += }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(SUMATERMINO) ∩ SIG(SUMATERMINO) = ¿? = { += } ∩ Ø = Ø . Cumple la regla #2*

TERMINO

TERMINO ➐ EXPRESIONNUM MASOPER

PRIM(TERMINO) = PRIM(EXPRESIONNUM)
={ round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length,
pow, log, tok_id, entero, flotante, +, - , (, }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASOPER

MASOPER ❼ ∑ **|**
OPER EXPRESIONNUM MASOPER

PRIM(MASOPER) = PRIM(OPER)
PRIM(MASOPER) = { *, / }

Regla #1: *Ø ∩ { *, / }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASOPER) ∩ SIG(MASOPER) = ¿? = { *, / } ∩ Ø = Ø . Cumple la regla #2.*

OPER

OPER ❼ *** | /**

PRIM (OPER) = { * , / }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

EXPRESIONBOOL

EXPRESIONBOOL ❼ **true |**
False |
Equal (EXPRESIONCAD , EXPRESIONCAD) |
tok_id

PRIM(**EXPRESIONBOOL**) = { true, false, equal,tok_id }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSTRUCCIÓN

INSTRUCCIÓN ❼ ∑ **|**
MASINSTRUCCION INSTRUCCIÓN

PRIM (INSTRUCCIÓN) = PRIM(MASINSTRUCCION)
PRIM (INSTRUCCIÓN) = { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}

Regla #1: *Ø ∩ { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(INSTRUCCION) ∩ SIG(INSTRUCCION) = ¿?.*

SIG(INSTRUCCIÓN) = { }, break} U PRIM(RETORNO) = { }, break, return }

PRIM(INSTRUCCION) ∩ SIG(INSTRUCCION)= { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} ∩ { }, break, return } = Ø . Cumple la regla #2

ASIGNACION

ASIGNACION ❼tok_id := EXPRESIONGENERAL

PRIM (ASIGNACION) = { tok_id }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASINSTRUCCION

MASINSTRUCCION ❼ **ASIGNACION ; |**

Tok_id ; | *Función de tipo void.*
INSIF |
INSWHILE |
INSFOR |
INSSWITCH |
INSDO |
INSCONSOLEWRITE ; |
INSCONSOLEREAD ; |
ABRIRARCHIVO ;

PRIM(MASINSTRUCCION) = PRIM(ASIGNACION) U { tok_id } U PRIM(INSIF) U PRIM(INSWHILE) U PRIM(INSFOR) U PRIM(INSSWITCH) U PRIM(INSDO) U PRIM(INSCONSOLEWRITE) U PRIM(INSCONSOLEREAD) U PRIM(ABRIRARCHIVO)

PRIM (MASINSTRUCCION) = { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}

Regla #1*= { tok_id }){ tok_id }){ if, while, for, switch, do, console.write, console.read, file.fopen} = { tok_id }*

Esta producción no es LL1 por lo que se aplicarán reglas semánticas, se tomará el tok_id de MASINSTRUCCION si es de tipo func.void en la tabla de símbolos, si es de tipo var. o arr.* en la tabla de símbolos se tomará el tok_id de ASIGNACION.*

EXPRESIONGENERAL

EXPRESIONGENERAL ❼ EXPRESION |
EXPRESIONARR

PRIM(EXPRESIONGENERAL) = PRIM(EXPRESION) U PRIM(EXPRESIONARR) = { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, entero, flotante, +, - , (, true, false, equal , Split , [}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

EXPRESIONARR

EXPRESIONARR ❼ **Split (EXPRESIONCAD, EXPRESIONCAD) |**
ARREGLO

PRIM (EXPRESIONARR) = { Split } U PRIM(ARREGLO)
PRIM(EXPRESIONARR) = { Split , [}

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

ARREGLO

ARREGLO ❼ **[ELEMENTOSARR]**

PRIM (ARREGLO) = { [}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

ELEMENTOSARR

ELEMENTOSARR ❼ **∑ |**
Flotante MASFLOTANTES |

Numero MASENTEROS |
Cadena MASCADENA |
BOOL MASBOOLEAN |
Carácter MASCARACTER

PRIM (ELEMENTOSARR) = { flotante, numero, cadena, true, false, carácter }

Regla #1: *Ø ∩ { flotante, numero, cadena, true, false, carácter } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(ELEMENTOSARR) ∩ SIG(ELEMENTOSARR) = ¿? = { flotante, numero, cadena, bool, carácter } ∩ {] } = Ø . Cumple la regla #2.*

BOOL

BOOL ❼ True | False

PRIM (BOOL) = { True, False }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASBOOLEAN

MASFLOTANTES ❼ ∑ **|**
, BOOL MASBOOLEAN|

PRIM (MASFLOTANTES) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASBOOLEAN) ∩ SIG(MASBOOLEAN) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASFLOTANTES

MASFLOTANTES ❼ ∑ **|**
, flotante MASFLOTANTES

PRIM (MASFLOTANTES) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASFLOTANTES) ∩ SIG(MASFLOTANTES) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASENTEROS

MASENTEROS ❼ ∑ **|**
, numero MASENTEROS

PRIM (MASENTEROS) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASENTEROS) ∩ SIG(MASENTEROS) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASCADENA

MASCADENA ❼ $\sum$ **|**
, cadena MASCADENA

PRIM (MASCADENA) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCADENA) ∩ SIG(MASCADENA) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASCARACTER

MASCARACTER ❼ $\sum$ **|**
, carácter MASCARACTER

PRIM (MASCARACTER) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCARACTER) ∩ SIG(MASCARACTER) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

INSIF

INSIF ❼ if PARENCONDICION LLAVEINSTRUCCION

PRIM (INSIF) = { if }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

PARENCONDICION

PARENCONIDICON ❼ (CONDICION)

PRIM (PARENCONDICION) = { (}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

LLAVEINSTRUCCION

LLAVEINSTRUCCION ❼ { INSTRUCCIÓN }

PRIM (LLAVEINSTRUCCION) = { { }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSWHILE

INSWHILE ❼ while PARENCONDICION LLAVEINSTRUCCION

PRIM (INSWHILE) = { while }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSFOR

INSFOR ❼ for (ASIGNACION ; CONDICION ; OPERACIONNUM) LLAVEINSTRUCCION

PRIM (INSFOR) = { for }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSSWITCH

INSSWITCH ❼ switch tok_id { CASE } *tok_id debe ser variable*

PRIM (INSSWITCH) = { switch }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CASE

CASE ❼ case PARENCONDICION { INSTRUCCIÓN break ; } MASCASE

PRIM (CASE) = { case }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASCASE

MASCASE ❼ case PARENCONDICION { INSTRUCCIÓN break ; } MASCASE |
default { INSTRUCCIÓN break ; }

PRIM (MASCASE) = { case, default }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSDO

INSDO ❼ do LLAVEINSTRUCCION while PARENCONDICION

PRIM (INSDO) = { do }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSCONSOLEWRITE

INSCONSOLEWRITE ❼ console.write (EXPRESIONCAD VARIABLESWRITE)

PRIM (INSCONSOLEWRITE) = { console.write }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

VARIABLESWRITE

VARIABLESWRITE ❼ ∑ |
, tok_id VARIABLESWRITE
PRIM (VARIABLESWRITE) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(VARIABLESWRITE) ∩ SIG(VARIABLESWRITE) = ¿? = { , } ∩ {) }= Ø . Cumple la regla #2.*

INSCONSOLEREAD

INSCONSOLEREAD ➐ console.read (CADVAR , & tok_id)

PRIM (INSCONSOLEREAD) = { console.read }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CADVAR

CADVAR ➐ %d | %c | %f | %s

PRIM (CADVAR) = { %d, %c, %f, %s }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

ABRIRARCHIVO

ABRIRARCHIVO ➐ file.fopen (DATOCAD , TIPOFILE) INSTRUCCIÓN file.fclose (tok_id)

PRIM (ABRIRARCHIVO) = { file.fopen }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

TIPOFILE

TIPOFILE ➐ " MODOFILE "

PRIM (TIPOFILE) = { " }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MODOFILE

MODOFILE ➐ a | r

PRIM(MODOFILE) = { a , r }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CONDICION

CONDICION ➐ CONDICIONGENERAL MASCONDICIONES

PRIM (CONDICION) = PRIM(CONDICIONGENERAL)

PRIM(CONDICION) = { true, false, equal, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (,substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CONDICIONGENERAL

CONDICIONGENERAL ➐ EXPRESIONBOOL | EXPRESIONNUM COMPARACION EXPRESIONNUM | EXPRESIONCAD COMPAGENERAL EXPRESIONCAD

PRIM(CONDICIONGENERAL) = PRIM(EXPRESIONBOOL) U PRIM(EXPRESIONNUM) U PRIM(EXPRESIONCAD)

={ true, false, equal, tok_id } U {round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante,+, - , (} U{ substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

= { true, false, equal, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (,substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *={ true, false, equal }){round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante,+, - , (}){ substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets } = { tok_id }*

Esta producción no es LL1, Sin embargo pueden aplicarse reglas semánticas. Tomará el camino de EXPRESIONNUM si tok_id es de tipo func.int, func.float, var.int, var.float, sino tomará el camino de EXPRESIONCAD, si tok_id en la tabla de símbolos es func.char, func.str, var.char, var.str, si tok_id es de tipo func.bool o var.bool entonces se ira a EXPRESIONBOOL

MASCONDICIONES

MASCONDICIONES ❼ ∑ **|**
|| CONDICION |
&& CONDICION

PRIM (MASCONDICIONES) = { ||, && }

Regla #1: *Ø ∩ { ||, && } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCONDICIONES) ∩ SIG(MASCONDICIONES) = ¿? = { ||, && } ∩ Ø = Ø . Cumple la regla #2.*

COMPARACION

COMPARACION ❼ **COMPAGENERAL |**
COMPANUM

PRIM (COMPARACION) = PRIM (COMPAGENERAL) U PRIM(COMPANUM)
PRIM (COMPARACION) = { ==, ¡= , < , >, >=, <= }

COMPAGENERAL

COMPAGENERAL ❼ == | ¡=

PRIM (COMPAGENERAL) = { == , ¡= }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

COMPANUM

COMPANUM ❼ < | > | >= | <=

PRIM(COMPANUM) = { < , >, >=, <= }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

PROGRAMA

PROGRAMA → main { BLOQUE } FUNCINST

PRIM (PROGRAMA) = { main }

Regla #1 : *Cumple con la regla #1 y no produce la palabra vacía*

Función Instrucción → FUNCINST

FUNCINST ❼ ∑ **|**
FUNCION tok_id CONJUNVAR { DECLARACION INSTRUCCIÓN RETORNO }

PRIM (FUNCINST) = PRIM(FUNCION)

PRIM (FUNCINST) = { func.int, func.char, func.str, func.bool, func.float }

Regla #1 : *Ø ∩ PRIM(FUNCION) = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(FUNCINST) ∩ SIG(FUNCINST) = ¿?*

={ func.int, func.char, func.str, func.bool, func.float } ∩ Ø = Ø Cumple con la regla #2

DECLARACION

DECLARACION ❼ ∑ **|**
VARIABLE tok_id ;

PRIM (DECLARACION) = PRIM(VARIABLE)

PRIM(DECLARACION) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1: *Ø ∩ PRIM(DECLARACION) = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(DECLARACION) ∩ SIG(DECLARACION) = ¿?*

SIG(DECLARACION) = PRIM(INSTRUCCIÓN) U PRIM(RETORNO) U PRIM(FUNCIONDECLA)

= { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} U { return } U { func.int, func.char, func.str, func.bool, func.float }

PRIM(DECLARACION) ∩ SIG(DECLARACION) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file } ∩ { tok_id, if, while, switch, do, console.write, console.read, file.fopen, return, func.int, func.char, func.str, func.bool, func.float } = Ø . Cumple con la regla #2.

RETORNO

RETORNO ❼ ∑ **|**
return EXPRESION ;

PRIM(RETORNO) = { return }

Regla #1 : *Ø ∩ { return } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(RETORNO) ∩ SIG(RETORNO) = ¿? = { return }∩ { } } = Ø. Cumple con la regla #2.*

VARIABLE

VARIABLE ❼ arr.int | arr.bool | arr.char | arr.str | arr.float |
var.int | var.bool | var.char | var.str | var.float | var.file

PRIM (VARIABLE) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1 : *Cumple con la regla #1 y no produce la palabra vacía*

FUNCION

FUNCION → **func.int | func.bool | func.char | func.str | func.float | func.void**

PRIM(FUNCION) = { func.int, func.char, func.str, func.bool, func.float, func.void }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía.*

CONJUNVAR

CONJUNVAR → (VARIABLE tok_id MASCONJUNVAR)

PRIM(CONJUNVAR) = { (}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía.*

MASCONJUNVAR

MASCONJUNVAR ❼ **∑ |**
, VARIABLE tok_id

PRIM (MASCONJUNVAR) = { , }***

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCONJUNVAR) ∩ SIG(MASCONJUNVAR) = ¿? = { , } ∩ {) }= Ø . Cumple la regla #2.*

BLOQUE

BLOQUE → **DECLARACION FUNCIONDECLA INSTRUCCIÓN**

PRIM(BLOQUE) = PRIM(DECLARACION) U PRIM(FUNCIONDECLA) U PRIM(INSTRUCCIÓN)

PRIM(BLOQUE) = { arr.int, arr.bool, arr.char, arr.str, arr.float, var.int, var.bool, var.char, var.str, var.float, var.file }

Regla #1 : *Cumple con la regla #1 y no produce la palabra vacía*

FUNCIONDECLA

FUNCIONDECLA → **∑ | FUNCION tok_id CONJUNVAR ; FUNCIONDECLA**

PRIM(FUNCIONDECLA) = PRIM(FUNCION)

PRIM(FUNCIONDECLA) = { func.int, func.char, func.str, func.bool, func.float }

Regla #1: *Ø ∩ { func.int, func.char, func.str, func.bool, func.float } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(FUNCIONDECLA) ∩ SIG(FUNCIONDECLA) = ¿?*

= { func.int, func.char, func.str, func.bool, func.float } ∩ PRIM(INSTRUCCIÓN) = Ø .

= { func.int, func.char, func.str, func.bool, func.float } ∩ { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} = Ø . Cumple la regla #2.

EXPRESION

EXPRESION → EXPRESIONCAD | EXPRESIONNUM | EXPRESIONBOOL

PRIM (EXPRESION) = PRIM(EXPRESIONCAD) U PRIM(EXPRESIONNUM) U PRIM(EXPRESIONBOOL)

= { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets } U { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (} U { true, false, equal,tok_id }
= { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, entero, flotante, +, - , (, true, false, equal, tok_id }

Regla # 1: Ø) PRIM(EXPRESIONCAD)) PRIM(EXPRESIONNUM) PRIM(EXPRESIONBOOL) = { tok_id }
Expresion no es LL1, Sin embargo se puede solucionar aplicando reglas semánticas. Se tomará el tok_id de EXPRESIONCAD si el identificador en la tabla de símbolos es de tipo func.str, func.char, var.char o var.str. Y el tok_id de EXPRESIONNUM se tomará si en la tabla de símbolos es de tipo func.int, func.float, var.int o var.float. Si es var.bool o func.bool se tomará EXPRESIONBOOL

EXPRESIONCAD

EXPRESIONCAD ❼ **FUNSUBSTRING |**
FUNCONCAT |
FUNREPLACE |
DATOCAD |
Tok_ id | *función de tipo Cadena o carácter*
file.scanf (tok_id)
file.fgets (tok_id)

PRIN (EXPRESIONCAD) = PRIM(FUNCSUBSTRING) U PRIM(FUNCONCAT) U PRIM(FUNREPLACE) U PRIM(DATOCAD) U { tok_id, file.scanf, file.fgets }

PRIM (EXPRESIONCAD) = { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *{ substring } ∩ { concat } ∩ { replace } ∩ { tok_id, cadena, carácter } ∩ { tok_id, file.scanf, file.fgets } = { tok_id }*

Esta producción no es LL1, sin embargo puede corregirse aplicando reglas semánticas, se tomará el tok_id de ExpresionCad si en la tabla de símbolos es una func.str o func.char, en cambio si en la tabla de símbolos es un var.str o un var.char se tomará el camino de DATOCAD.

FUNSUBSTRING

FUNSUBSTRING → substring (EXPRESIONCAD , EXPRESIONNUM , EXPRESIONNUM)

PRIM(FUNSUBSTRING) = { substring }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

FUNCONCAT

FUNCONCAT → concat (EXPRESIONCAD , EXPRESIONCAD)

PRIM(FUNCONCAT) = { concat }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

FUNREPLACE

FUNREPLACE → replace (EXPRESIONCAD , EXPRESIONCAD , EXPRESIONCAD)

PRIM(FUNREPLACE) = { replace }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

DATOCAD

DATOCAD ❼ IDENTIFICADOR |
Cadena |
Carácter

PRIM(DATOCAD) = PRIM(IDENTIFICADOR) U { cadena, caracter }

PRIM(DATOCAD) = { tok_id, cadena, carácter }

Regla #1: *PRIM(IDENTIFICADOR) ∩ { cadena, caracter} = Ø , Cumple la regla 1 y no produce la palabra vacía.*

IDENTIFICADOR

IDENTIFICADOR → tok_id IDENARR

PRIM(IDENTIFICADOR) = { tok_id }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

IDENARR

IDENARR ❼ ∑ |
[VALORARR]

PRIM(IDENARR) = { [}

Regla #1: Ø ∩ { [} = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.

Regla #2: PRIM(IDENARR) ∩ SIG(IDENARR) = ¿? = { [} ∩ Ø = Ø . Cumple la regla #2.

VALORARR

VALORARR ❼ tok_id | *Variable de tipo entero*
entero

PRIM (VALORARR) = { tok_id, entero }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

EXPRESIONNUM

EXPRESIONNUM ❼ **round (EXPRESIONNUM) |**
Sin (EXPRESIONNUM) |
Cos (EXPRESIONNUM) |
Arcsin (EXPRESIONNUM) |
Arccos (EXPRESIONNUM) |
Arctan (EXPRESIONNUM) |
Tan (EXPRESIONNUM) |
Sqrt (EXPRESIONNUM) |
Length (EXPRESIONCAD) |
Pow (EXPRESIONNUM , EXPRESIONNUM) |
Log (EXPRESIONNUM , EXPRESIONNUM) |
DATONUM |
OPERACIONNUM |
(EXPRESIONNUM) |
Tok_id *función entero o flotante*

PRIM (EXPRESIONNUM) = { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log } U PRIM(DATONUM) U PRIM(OPERACIONNUM) U { (, tok_id}

PRIM(EXPRESIONNUM) = { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (, }

Regla #1: { round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log } *)* { tok_id, entero, flotante } *)* { +, - } *)*{ *(, tok_id } = { tok_ id }*

Esta producción no es LL1 sin embargo se puede solucionar con reglas semánticas, se tomará el camino del tok_id de EXPRESIONNUM si en la tabla de símbolos es una func.int o func.float, de lo contrato si en la tabla de símbolos es un var.int o un var.float se tomará el camino de DATONUM.

DATONUM

DATONUM ❼ **IDENTIFICADOR SUMATERMINO|** *Variable de tipo entero o flotante*
Entero |
Flotante

PRIM(DATONUM) = PRIM(IDENTIFICADOR) U { entero, flotante }
PRIM(DATONUM) = { tok_id, entero, flotante }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

OPERACIONNUM

OPERACIONNUM ❼ **OPERATERMINO TERMINO MASOPERACIONNUM**

PRIM(OPERACIONNUM) = PRIM (OPERATERMINO)
PRIM(OPERACIONNUM)= { +, - }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASOPERACIONNUM

MASOPERACIONNUM ❼ ∑ | OPERATERMINO TERMINO MASOPERACIONNUM

PRIM(MASOPERACIONNUM) = PRIM(OPERATERMINO) = { +, - }

Regla #1: *Ø ∩ { +, - }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASOPERACIONNUM) ∩ SIG(MASOPERACIONNUM) = ¿? = { +, - } ∩ Ø = Ø . Cumple la regla #2.*

OPERATERMINO

OPERATERMINO ❼ + | -

PRIM (OPERATERMINO) = { + , - }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

SUMATERMINO

SUMATERMINO ❼ ∑ |
+ = EXPRESIONNUM

PRIM (SUMATERMINO) = { += }

Regla #1: *Ø ∩ { += }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(SUMATERMINO) ∩ SIG(SUMATERMINO) = ¿? = { += } ∩ Ø = Ø . Cumple la regla #2.*

TERMINO

TERMINO ❼ EXPRESIONNUM MASOPER

PRIM(TERMINO) = PRIM(EXPRESIONNUM)
={ round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length,
pow, log, tok_id, entero, flotante, +, - , (, }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASOPER

MASOPER ❼ ∑ |
OPER EXPRESIONNUM MASOPER

PRIM(MASOPER) = PRIM(OPER)
PRIM(MASOPER) = { *, / }

Regla #1: *Ø ∩ { *, / }= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASOPER) ∩ SIG(MASOPER) = ¿? = { *, / } ∩ Ø = Ø . Cumple la regla #2.*

OPER

OPER ❼ * | /

PRIM (OPER) = { * , / }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

EXPRESIONBOOL

EXPRESIONBOOL ❼ **true |**
False |
Equal (EXPRESIONCAD , EXPRESIONCAD) |
tok_id

PRIM(**EXPRESIONBOOL**) = { true, false, equal,tok_id }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSTRUCCIÓN

INSTRUCCIÓN ❼ ∑ **|**
MASINSTRUCCION INSTRUCCIÓN

PRIM (INSTRUCCIÓN) = PRIM(MASINSTRUCCION)

PRIM (INSTRUCCIÓN) = { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}

Regla #1: *Ø ∩ { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}= Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(INSTRUCCION) ∩ SIG(INSTRUCCION) = ¿?.*

SIG(INSTRUCCIÓN) = { }, break} U PRIM(RETORNO) = { }, break, return }

PRIM(INSTRUCCION) ∩ SIG(INSTRUCCION)= { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen} ∩ { }, break, return } = Ø . Cumple la regla #2

ASIGNACION

ASIGNACION ❼tok_id := EXPRESIONGENERAL

PRIM (ASIGNACION) = { tok_id }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASINSTRUCCION

MASINSTRUCCION ❼ **ASIGNACION ; |**
Tok_id ; | *Función de tipo void.*
INSIF |
INSWHILE |
INSFOR |
INSSWITCH |
INSDO |
INSCONSOLEWRITE ; |
INSCONSOLEREAD ; |
ABRIRARCHIVO ;

PRIM(MASINSTRUCCION) = PRIM(ASIGNACION) U { tok_id } U PRIM(INSIF) U PRIM(INSWHILE) U PRIM(INSFOR) U PRIM(INSSWITCH) U PRIM(INSDO) U PRIM(INSCONSOLEWRITE) U PRIM(INSCONSOLEREAD) U PRIM(ABRIRARCHIVO)

PRIM (MASINSTRUCCION) = { tok_id, if, while, for, switch, do, console.write, console.read, file.fopen}

Regla #1*= { tok_id }) { tok_id }) { if, while, for, switch, do, console.write, console.read, file.fopen} = { tok_id }*

Esta producción no es LL1 por lo que se aplicarán reglas semánticas, se tomará el tok_id de MASINSTRUCCION si es de tipo func.void en la tabla de símbolos, si es de tipo var. o arr.* en la tabla de símbolos se tomará el tok_id de ASIGNACION.*

EXPRESIONGENERAL

EXPRESIONGENERAL ❼ EXPRESION |
EXPRESIONARR

PRIM(EXPRESIONGENERAL) = PRIM(EXPRESION) U PRIM(EXPRESIONARR)
= { substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, entero, flotante, +, - , (, true, false, equal , Split , [}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

EXPRESIONARR

EXPRESIONARR ❼ Split (EXPRESIONCAD, EXPRESIONCAD) |
ARREGLO

PRIM (EXPRESIONARR) = { Split } U PRIM(ARREGLO)
PRIM(EXPRESIONARR) = { Split , [}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

ARREGLO

ARREGLO ❼ [ELEMENTOSARR]

PRIM (ARREGLO) = { [}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

ELEMENTOSARR

ELEMENTOSARR ❼ ∑ |
Flotante MASFLOTANTES |
Numero MASENTEROS |
Cadena MASCADENA |
BOOL MASBOOLEAN |
Carácter MASCARACTER

PRIM (ELEMENTOSARR) = { flotante, numero, cadena, true, false, carácter }

Regla #1: *Ø ∩ { flotante, numero, cadena, true, false, carácter } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(ELEMENTOSARR) ∩ SIG(ELEMENTOSARR) = ¿? = { flotante, numero, cadena, bool, carácter } ∩ {] } = Ø . Cumple la regla #2.*

BOOL

BOOL ❼ True | False

PRIM (BOOL) = { True, False }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASBOOLEAN

MASFLOTANTES ❼ ∑ **|**
, BOOL MASBOOLEAN|

PRIM (MASFLOTANTES) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASBOOLEAN) ∩ SIG(MASBOOLEAN) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASFLOTANTES

MASFLOTANTES ❼ ∑ **|**
, flotante MASFLOTANTES

PRIM (MASFLOTANTES) = { , }

Regla #1: Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.

Regla #2: PRIM(MASFLOTANTES) ∩ SIG(MASFLOTANTES) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.

MASENTEROS

MASENTEROS ❼ ∑ **|**
, numero MASENTEROS

PRIM (MASENTEROS) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASENTEROS) ∩ SIG(MASENTEROS) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASCADENA

MASCADENA ❼ ∑ **|**
, cadena MASCADENA

PRIM (MASCADENA) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCADENA) ∩ SIG(MASCADENA) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

MASCARACTER

MASCARACTER ❼ ∑ **|**
, carácter MASCARACTER

PRIM (MASCARACTER) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCARACTER) ∩ SIG(MASCARACTER) = ¿? = { , } ∩ Ø = Ø . Cumple la regla #2.*

INSIF

INSIF ❼ if PARENCONDICION LLAVEINSTRUCCION

PRIM (INSIF) = { if }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

PARENCONDICION

PARENCONIDICON ❼ (CONDICION)

PRIM (PARENCONDICION) = { (}

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

LLAVEINSTRUCCION

LLAVEINSTRUCCION ❼ { INSTRUCCIÓN }

PRIM (LLAVEINSTRUCCION) = { { }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSWHILE

INSWHILE ❼ while PARENCONDICION LLAVEINSTRUCCION

PRIM (INSWHILE) = { while }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSFOR

INSFOR ❼ for (ASIGNACION ; CONDICION ; OPERACIONNUM) LLAVEINSTRUCCION

PRIM (INSFOR) = { for }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSSWITCH

INSSWITCH ❼ switch tok_id { CASE } *tok_id debe ser variable*

PRIM (INSSWITCH) = { switch }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CASE

CASE ❼ case PARENCONDICION { INSTRUCCIÓN break ; } MASCASE

PRIM (CASE) = { case }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

MASCASE

MASCASE ❼ case PARENCONDICION { INSTRUCCIÓN break ; } MASCASE |
default { INSTRUCCIÓN break ; }

PRIM (MASCASE) = { case, default }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSDO

INSDO ❼ do LLAVEINSTRUCCION while PARENCONDICION

PRIM (INSDO) = { do }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

INSCONSOLEWRITE

INSCONSOLEWRITE ➐ console.write (EXPRESIONCAD VARIABLESWRITE)

PRIM (INSCONSOLEWRITE) = { console.write }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

VARIABLESWRITE

VARIABLESWRITE ➐ $\sum$ **|**
, tok_id VARIABLESWRITE

PRIM (VARIABLESWRITE) = { , }

Regla #1: *Ø ∩ { , } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(VARIABLESWRITE) ∩ SIG(VARIABLESWRITE) = ¿? = { , } ∩ {) }= Ø . Cumple la regla #2.*

INSCONSOLEREAD

INSCONSOLEREAD ➐ console.read (CADVAR , & tok_id)

PRIM (INSCONSOLEREAD) = { console.read }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CADVAR

CADVAR ➐ %d | %c | %f | %s

PRIM (CADVAR) = { %d, %c, %f, %s }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

ABRIRARCHIVO

ABRIRARCHIVO ➐ file.fopen (DATOCAD , TIPOFILE)

PRIM (ABRIRARCHIVO) = { file.fopen }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

TIPOFILE

TIPOFILE ➐ “ MODOFILE “

PRIM (TIPOFILE) = { “ }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

MODOFILE

MODOFILE ➐ a | r

PRIM(MODOFILE) = { a , r }

Regla #1: Cumple con la regla #1 y no produce la palabra vacía

CONDICION

CONDICION ➐ CONDICIONGENERAL MASCONDICIONES

PRIM (CONDICION) = PRIM(CONDICIONGENERAL)

PRIM(CONDICION) = { true, false, equal, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (,substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

CONDICIONGENERAL

CONDICIONGENERAL ❼ **EXPRESIONBOOL |**
EXPRESIONNUM COMPARACION EXPRESIONNUM |
EXPRESIONCAD COMPAGENERAL EXPRESIONCAD

PRIM(CONDICIONGENERAL) = PRIM(EXPRESIONBOOL) U PRIM(EXPRESIONNUM) U PRIM(EXPRESIONCAD)

={ true, false, equal, tok_id } U {round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante,+, - , (} U{ substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

= { true, false, equal, round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante, +, - , (,substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets }

Regla #1: *={ true, false, equal }) {round, sin, cos, arcsin, arccos, arctan, tan, sqrt, length, pow, log, tok_id, entero, flotante,+, - , (}){ substring, concat, replace, tok_id,cadena, carácter, file.scanf, file.fgets } = { tok_id }*

Esta producción no es LL1, Sin embargo pueden aplicarse reglas semánticas. Tomará el camino de EXPRESIONNUM si tok_id es de tipo func.int, func.float, var.int, var.float, sino tomará el camino de EXPRESIONCAD, si tok_id en la tabla de símbolos es func.char, func.str, var.char, var.str, si tok_id es de tipo func.bool o var.bool entonces se ira a EXPRESIONBOOL

MASCONDICIONES

MASCONDICIONES ❼ **∑ |**
|| CONDICION |
&& CONDICION

PRIM (MASCONDICIONES) = { ||, && }

Regla #1: *Ø ∩ { ||, && } = Ø , Como produce la palabra vacía (ε) se aplicará también la regla #2.*

Regla #2: *PRIM(MASCONDICIONES) ∩ SIG(MASCONDICIONES) = ¿? = { ||, && } ∩ Ø = Ø . Cumple la regla #2.*

COMPARACION

COMPARACION ❼ **COMPAGENERAL |**
COMPANUM

PRIM (COMPARACION) = PRIM (COMPAGENERAL) U PRIM(COMPANUM)

PRIM (COMPARACION) = { ==, ¡= , < , >, >=, <= }

COMPAGENERAL

COMPAGENERAL ❼ == | ¡=

PRIM (COMPAGENERAL) = { == , ¡= }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*

COMPANUM

COMPANUM ❼ < | > | >= | <=

PRIM(COMPANUM) = { < , >, >=, <= }

Regla #1: *Cumple con la regla #1 y no produce la palabra vacía*