



Universidad Centroamericana  
"José Simeón Cañas"  
Depto. Electrónica e  
Informática

# Armus

Lenguaje de Programación  
Orientado a Objetos

## ► Documento Técnico

Manual de Referencia y Ayuda

**Materia:** Compiladores

**Catedrático:** Lic. José Enmanuel Amaya Araujo

**Desarrollado por grupo ACL:**

- » Néstor Santiago Aldana Rodríguez
- » Bárbara Stefany Aparicio Bermúdez
- » Katherine Stephanie Cabrera Blanco
- » Diana Marcela López Rosales
- » Carlos Miguel López Loarca

UCA | Diciembre 2016

```
JNIEXPORT jobjectArray JNICALL Java_armus_lib_parser_Parser_00028
    (JNIEnv* env, jobject obj, jobjectArray jsLsFile) {
    //cantidad de archivos mandados
    int cant = (*env)->GetArrayLength(env, jsLsFile);
    char **lsfiles; // Lista de archivos CONVERTIDAS a c
    void **jstrings;
    //reservado espacio para la conversion
    lsfiles = (char **) malloc(sizeof (const char *) * cant);
    jstrings = (char **) malloc(sizeof (jstring *) * cant);
    int i;
    //Conversion
    for (i = 0; i < cant; i++) {
        jstring string = (jstring) ((*env)->GetObjectArrayElement(env, jsLsFile, i));
        const char *file = (*env)->GetStringUTFChars(env, string, 0);
        lsfiles[i] = file;
        jstrings[i] = &string;
    }

    tabla.izq = NULL;
    tabla.dch = NULL;
    tabla.valor = NULL;

    (pasada1(lsfiles, cant)) {
        if (pasada2(lsfiles, cant)) {
            return NULL; // Sin errores
        }
    }

    //ay errores;

    (env)->NewObjectArray(env, primerError,
        (*env)->FindClass(env, "java/lang/String"), NULL);

    for (j = 0; j < primerError ; j++){
        jobjectArrayElement(env, errores, j, (*env)->FindClass(env, "java/lang/String"), NULL);
    }
}
```

# Índice

<b>A.</b> Descripción del diseño del lenguaje y de sus posibles aplicaciones.....	3
<b>B.</b> Palabras Reservadas.....	7
<b>C.</b> Expresión regular para los identificadores.....	8
<b>D.</b> Expresiones regulares para números enteros y números reales.....	8
<b>E.</b> Expresiones regulares para cadenas y caracteres.....	8
<b>F.</b> Lista de operadores y caracteres especiales .....	9
<b>G.</b> Forma de construcción de comentarios en el lenguaje.....	9
<b>H.</b> Un ejemplo de programa para escribir “Hola mundo” en el lenguaje .....	9
<b>I.</b> Configuración para la generación de la librería de JNI para el IDE y Analizador Lexicográfico en Netbeans sobre Linux Ubuntu 14.04.....	10
<b>J.</b> Lista de Tokens utilizados.....	13
<b>K.</b> Diagramas de Sintaxis .....	13
<b>L.</b> Gramática Formal .....	26
<b>M.</b> Aplicación de la Regla de Primero .....	30
<b>N.</b> Aplicación de las Reglas de Primero y Siguiente.....	35
<b>O.</b> Reglas Semánticas Extras Aplicadas .....	46
<b>P.</b> Diagrama de Dependencias.....	47
<b>Q.</b> Estrategias para la estabilización de errores sintácticos y semánticos .....	49
<b>R.</b> Estrategia para el Manejo de Código Intermedio (Múltiples Archivos).....	49
<b>S.</b> Diseño del Código Intermedio .....	50

## A. Descripción del diseño del lenguaje y de sus posibles aplicaciones.

El Lenguaje de programación se basa en la ejecución de algoritmos en pseudo-código en el lenguaje español, lo cual fortalece el práctico entendimiento en las personas que se inician en el ámbito de la programación, y además brinda una de una herramienta más clara para el aprendizaje del paradigma de programación orientada a objetos.

### Origen del lenguaje Armus

El nombre del lenguaje que se encuentra en la etapa de diseño e identificación lexicográfica ha sido nombrado como **Armus**, dicho nombre tiene origen en el nombre de una estrella llamada Eta Capricorni<sup>1</sup> (también conocida como Armus) es una estrella binaria en la constelación de Capricornio.

### Estructura y sintaxis

A continuación, se define toda la estructura y sintaxis para la ejecución de algoritmos.

- Paradigma del lenguaje: Programación Orientada a Objetos (POO)
- Lenguaje sensitivo a mayúsculas y minúsculas (case sensitive).

<b>Manejo de múltiples archivos de código fuente</b>	<p>El lenguaje soporta la inclusión de múltiples archivos, en uno o más archivos que conforman el código fuente completo del programa escrito en el lenguaje Armus.</p> <p>El parámetro de configuración <b>MAX_FILE</b> define el número máximo de archivos que pueden existir conteniendo todo el código fuente del programa.</p> <p>El parámetro de configuración <b>MAX_NAME_FILE</b> define el número máximo de caracteres que puede tener el nombre o ruta del archivo a incluir.</p> <p><u>Estructura:</u> <b>incluir</b> &lt;ruta o nombre del archivo a ser incluido&gt;;</p> <p><u>Ejemplos:</u> <b>incluir</b> "/home/mike/Documentos/proyecto/Principal.acl"; <b>incluir</b> "../Documentos/proyecto/Fuentes.acl"; <b>incluir</b> "Calculos.acl";</p> <p><b>NOTA:</b> La ruta puede ser absoluta o relativa en Linux exclusivamente.</p>
<b>Manejo de variables</b>	<p><u>Estructura:</u> &lt;tipo de dato&gt; &lt;variable alfanumérica&gt;;</p> <p><u>Ejemplos de sintaxis:</u> <b>entero</b> n1; <b>byte</b> num, contador, i; <b>real</b> temp;</p>
<b>Tipos de datos</b>	<ol style="list-style-type: none"><li>1. <b>vacio</b> (equivalente a void)</li><li>2. <b>booleano</b> (verdadero o falso)</li></ol>

<sup>1</sup> [https://es.wikipedia.org/wiki/Eta\\_Capricorni](https://es.wikipedia.org/wiki/Eta_Capricorni)

	3. <b>cadena</b> (cadena de caracteres) 4. <b>caracter</b> (equivalente a char), 2 Bytes 5. <b>byte</b> (-127 a 128), 1 Byte 6. <b>entero</b> (equivalente a int), 4 Bytes 7. <b>real</b> (equivalente float), 4 Bytes 8. <b>Objeto</b> (equivalente a Object) 9. <b>Archivo</b> (equivalente a File)
<b>Instrucciones de Selección</b>	<pre> si[&lt;condición&gt;]{     &lt;sentencias&gt;; }sino{     &lt;sentencias&gt;; } </pre>
	<pre> si[&lt;condición&gt;, &lt;valor verdadero&gt;, &lt;valor falso&gt;]; </pre> <p><b>NOTA:</b> la opción verdadera y falsa no se pueden omitir y deben ser valores.</p>
	<pre> probar[&lt;condición&gt;]{     caso 1: &lt;sentencia1&gt;;            &lt;sentencia2&gt;;            romper;     caso 2: &lt;sentencia1&gt;;            &lt;sentencia2&gt;;            romper;     defecto: &lt;sentencia1&gt;;             &lt;sentencia2&gt;;             romper; } </pre>
<b>Instrucciones Iterativas</b>	<pre> mientras[&lt;condición&gt;]{ } /* el equivalente a while */ </pre>
	<pre> para[&lt;condición&gt;, &lt;valor inicial&gt;, &lt;incremento&gt;]{ } /* el equivalente a for */ </pre>
	<pre> hacer{ }mientras[&lt;condición&gt;; /* el equivalente a do-while */ </pre>
	<pre> paraCada[&lt;tipo&gt; &lt;variable&gt;, &lt;arreglo/lista&gt;]{ } /* el equivalente a for-each */ </pre>
<b>Instrucciones de entrada y salida</b>	<p><u>Sintaxis de entrada:</u></p> <p><b>Sistema.obtenerEntero</b>[&lt;var&gt;]; //lee un entero</p> <p><b>Sistema.obtenerReal</b>[&lt;var&gt;]; //lee un núm. real</p> <p><b>Sistema.obtenerCadena</b>[&lt;var&gt;]; //lee una cadena</p> <p><b>Sistema.obtenerCaracter</b>[&lt;var&gt;]; //lee un carácter</p>

	<pre> &lt;objeto&gt;.obtenerEntero[&lt;var&gt;]; &lt;objeto&gt;.obtenerReal[&lt;var&gt;]; &lt;objeto&gt;.obtenerCadena[&lt;var&gt;]; &lt;objeto&gt;.obtenerCaracter[&lt;var&gt;]; </pre>
	<div> <div> <p><u>Sintaxis de salida:</u>  <b>Sistema.mostrar</b>["hola"];   <b>Sistema.mostrar</b>["%e + 2", valor];   &lt;objeto&gt;.mostrar[];</p> </div> <div> <p>%e -&gt; entero  %r -&gt; real  %c -&gt; carácter  %s -&gt; cadena</p> </div> </div>
<b>Sub-Algoritmos</b>	<pre> <b>publica</b> &lt;función&gt;[&lt;parámetros entrada&gt;] &lt;tipo salida&gt; { /* puede no tener parámetros de entrada y el tipo de salida puede ser vacío */ <b>retornar</b> &lt;valor de retorno&gt;; }  <b>privada</b> &lt;función&gt;[&lt;parámetros entrada&gt;] &lt;tipo salida&gt; { /* puede no tener parámetros de entrada o el tipo de salida puede ser vacío */ <b>retornar</b> &lt;valor de retorno&gt;; }  <b>NOTA:</b> Los procedimientos pueden o no recibir parámetros de entrada (opcional), pero <u>NO retornan nada</u>.  <b>publica</b> &lt;procedimiento&gt;[&lt;parámetros entrada&gt;]{ } <b>privada</b> &lt;procedimiento&gt;[&lt;parámetros entrada&gt;]{ }  <b>NOTA 2:</b> En ambos casos se utilizan sobre objetos.  &lt;objeto&gt;.&lt;función&gt;[&lt;parámetros entrada&gt;] &lt;tipo salida&gt;; &lt;objeto&gt;.&lt;procedimiento&gt;[&lt;parámetros entrada&gt;]; &lt;objeto&gt;.&lt;procedimiento&gt;[];  <u>Ejemplo:</u> Alumno.ObtenerNombre[]; </pre>
<b>Paso de parámetros por valor y por referencia</b>	<p><u>Por valor</u></p> <pre> &lt;objeto&gt;.&lt;función&gt;[11, 24]; .. &lt;función&gt;[entero n1, entero n2]{ } </pre> <p><u>Por referencia</u></p> <pre> &lt;objeto&gt;.&lt;función&gt;[~&lt;parámetro&gt;]; </pre> <p>Si el parámetro es enviado por referencia, se antepone el símbolo ~</p>

	<pre> .. entero n1 = 11; &lt;objeto&gt;.&lt;función&gt;[~n1] .. &lt;función&gt;[entero *n]{     *n = 24; } .. Sistema.mostrar[n1]; //muestra: 24 </pre> <p>Si el parámetro es recibido por referencia en la definición de la función se coloca * antes del nombre del parámetro.</p> <p><b>NOTA:</b> Se ha pasado por la referencia el parámetro enviado a la función, es decir, que todo lo que se cambie al parámetro dentro de la función también se cambiará en el parámetro original (no se hace una copia del parámetro).</p>
<b>Manejo de Arreglos</b>	<pre> Arreglo&lt;tipo&gt; &lt;objeto arreglo&gt;;  &lt;objeto arreglo&gt;.agregar[&lt;elemento1&gt;]; &lt;objeto arreglo&gt;.agregar[&lt;elemento2&gt;];  &lt;objeto arreglo&gt;.obtener[&lt;elemento1&gt;]; &lt;objeto arreglo&gt;.obtener[&lt;elemento2&gt;];  &lt;objeto arreglo&gt;.cuantos[]; &lt;objeto arreglo&gt;.quitar[&lt;elemento&gt;]; </pre>
<b>Manejo Básico de Archivos</b> (apertura y volcado)	<pre> Archivo &lt;objeto&gt;;  &lt;objeto&gt;.abrir[&lt;ruta o nombre del archivo&gt;];  &lt;objeto&gt;.leerLinea[]; /*lee solo una línea del archivo*/  &lt;objeto&gt;.volcado[]; /*obtiene todo el contenido del archivo*/  &lt;objeto&gt;.cerrar[]; /*cierra el archivo*/ </pre>
<b>Funciones Predefinidas</b>	<pre> concatenar[&lt;cadena1&gt;, &lt;cadena2&gt;];  parteEntera[&lt;número real&gt;]; /*extrae la parte entera de un número real*/  comparar[&lt;cadena1&gt;, &lt;cadena2&gt;]; /*devuelve 1 si son iguales, 0 diferentes*/  mayor[&lt;num1&gt;, &lt;num2&gt;]; /*devuelve el mayor de ambos números */  menor[&lt;num1&gt;, &lt;num2&gt;]; /*devuelve el menor de ambos números */  esPar[&lt;numero&gt;]; /*devuelve 1 si es par, 0 si no lo es*/  decimalBin[&lt;número entero&gt;]; /*transforma de decimal a binario*/  potencia[&lt;número&gt;, &lt;exponente&gt;];  absoluto[&lt;numero&gt;]; /*obtiene el valor absoluto de un numero*/  modulo[&lt;entero1&gt;,&lt;entero2&gt;]; /*obtiene el residuo de una división*/  longitudCadena[&lt;cadena&gt;]; /*devuelve la longitud de una cadena*/ </pre>

	NOTA: todas están definidas en el objeto <b>Sistema</b>																																				
Operadores Aritméticos y Lógicos	<u>Lógicos</u>		<u>Orden de Prioridad</u>																																		
	<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>==</td><td>Igual</td></tr><tr><td>&lt;</td><td>menor</td></tr><tr><td>&lt;=</td><td>Menor o igual</td></tr><tr><td>&gt;</td><td>Mayor</td></tr><tr><td>&gt;=</td><td>Mayor o igual</td></tr><tr><td>&lt;&gt;</td><td>!=</td><td>Distinto</td></tr><tr><td>&amp;&amp;</td><td colspan="2">Operador Y</td></tr><tr><td>  </td><td colspan="2">Operador O</td></tr><tr><td>!</td><td colspan="2">Negación</td></tr></table>		Operador	Descripción	==	Igual	<	menor	<=	Menor o igual	>	Mayor	>=	Mayor o igual	<>	!=	Distinto	&&	Operador Y			Operador O		!	Negación		<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>()</td><td>Paréntesis</td></tr><tr><td>* , /</td><td>Multiplicación, División</td></tr><tr><td>+ , -</td><td>Suma, Resta</td></tr><tr><td>==,&lt;&gt;,&gt;,&lt;,&gt;=,&lt;=</td><td>Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual</td></tr></table>	Operador	Descripción	()	Paréntesis	* , /	Multiplicación, División	+ , -	Suma, Resta	==,<>,>,<,>=,<=	Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual
	Operador	Descripción																																			
	==	Igual																																			
	<	menor																																			
	<=	Menor o igual																																			
	>	Mayor																																			
	>=	Mayor o igual																																			
	<>	!=	Distinto																																		
	&&	Operador Y																																			
	Operador O																																				
!	Negación																																				
Operador	Descripción																																				
()	Paréntesis																																				
* , /	Multiplicación, División																																				
+ , -	Suma, Resta																																				
==,<>,>,<,>=,<=	Igual, Distinto, Mayor, Menor, Mayor o igual, Menor o igual																																				
<u>Aritméticos</u>																																					
<table><tr><th>Operador</th><th>Descripción</th></tr><tr><td>+</td><td>Suma</td></tr><tr><td>-</td><td>Resta</td></tr><tr><td>*</td><td>Multiplicación</td></tr><tr><td>/</td><td>División</td></tr></table>			Operador	Descripción	+	Suma	-	Resta	*	Multiplicación	/	División																									
Operador	Descripción																																				
+	Suma																																				
-	Resta																																				
*	Multiplicación																																				
/	División																																				
Comentarios	// comentario de una sola línea  /* Comentario para párrafos. */																																				

## B. Palabras Reservadas

entero	caso	retornar	menor
byte	defecto	Arreglo	esPar
real	romper	agregar	decimalBin
vacio	mientras	obtener	pontencia
booleano	para	cuantos	absoluto
cadena	hacer	quitar	modulo
caracter	paraCada	abrir	longitudCadena
Objeto	Sistema	leerLinea	clase
Archivo	obtenerEntero	volcado	incluir
si	obtenerReal	cerrar	verdadero
sino	obtenerCadena	concatenar	false
probar	obtenerCaracter	parteEntera	obtenerBooleano
publica	mostrar	comparar	mayor
privada	local	principal	

## C. Expresión regular para los identificadores

- VARIABLES

identificador = (letra + símbolo)(letra + dígito + símbolo)\*

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {"\_"}

- CLASES (*Sub-conjunto de identificador de variable*)

identificador = letraMa(letra + dígito + símbolo)\*

Donde:

letraMa = {A, B, C, ..., Z}

letra = {a, b, ..., z, A, B, ..., Z}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {"\_"}

## D. Expresiones regulares para números enteros y números reales

- NÚMEROS ENTEROS & HEXADECIMALES

entero = dígito(dígito)\* + #(letra + dígito)\*

Donde:

dígito = {0,1,2,3,4,5,6,7,8,9}

letra = {A, B, C, D, E, F} (números Hexadecimal 10, 11, 12, 13, 14, 15)

dígito = {0,1,2,3,4,5,6,7,8,9}

Un número entero puede ser definido como un número Hexadecimal, Ejemplo: #A32FF (anteponiendo el símbolo #)

- NÚMEROS REALES

real = entero.entero

## E. Expresiones regulares para cadenas y caracteres

- CADENA

cadena = "(letra + dígito + símbolo)\*"

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {x / x es cualquier carácter diferente de letra, dígito o "("}

- CARÁCTER

caracter = '(letra + dígito + símbolo)'

Donde:

letra = {a, b, ..., ñ, ..., z, A, B, ..., Ñ, ..., Z, á, é, í, ó, ú, Á, É, Í, Ó, Ú}

dígito = {0,1,2,3,4,5,6,7,8,9}

símbolo = {x / x es cualquier carácter diferente de letra, dígito o "("}



## F. Lista de operadores y caracteres especiales

### Lógicos

Operador	Descripción
==	Igual
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
<>    !=	Distinto
&&	Operador Y
	Operador O
!	Negación

### Aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División

### Caracteres Especiales

Carácter	Descripción
~	Referencia
.	Punto
#	Inicio de Número Hexadecimal
;	Punto y coma
,	Coma
:	Dos puntos
=	Asignación
{	Llave de apertura
}	Llave de cierre
[	Corchete de apertura
]	Corchete de cierre
(	Paréntesis de apertura
)	Paréntesis de cierre

**NOTA:** En la composición de cadenas se agregan además todos los caracteres de la tabla ASCII

## G. Forma de construcción de comentarios en el lenguaje

Para la utilización de los comentarios se hará de igual forma que en lenguaje C y Java, para los comentarios de una sola línea se utiliza `//` , para los comentarios de párrafos `/* */`

### Ejemplos:

`// comentario de una sola línea`

```
/*
Comentario para párrafos.
*/
```

## H. Un ejemplo de programa para escribir “Hola mundo” en el lenguaje

Manejo de ámbito de variables y métodos: **pública y privada**

```
clase <nombre_clase>{
    privada entero variable;
    privada metodo[] {
        entero var;
        var = variable;
    }
}
```

Ejemplo (hola mundo):

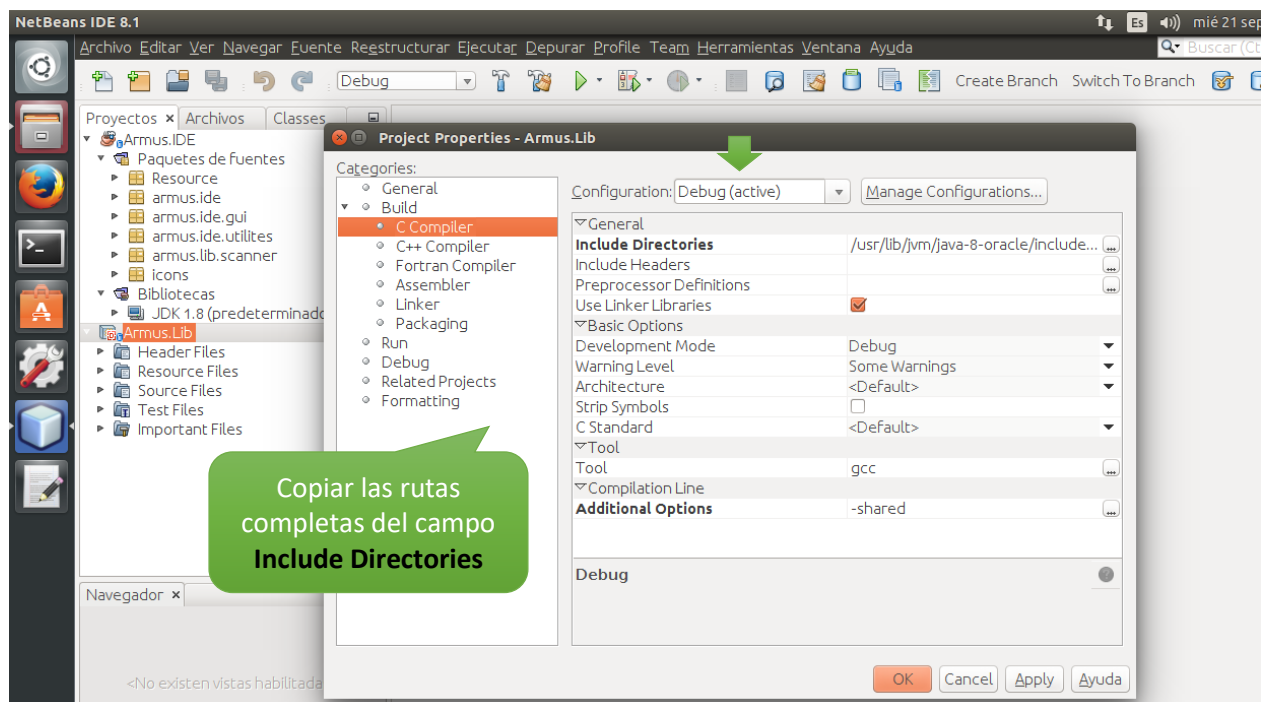
```
clase Hola{
    publica principal [] {
        Sistema.mostrar[“Hola mundo\n”];
    }
    /*comentario*/
}
```

Ejemplo:

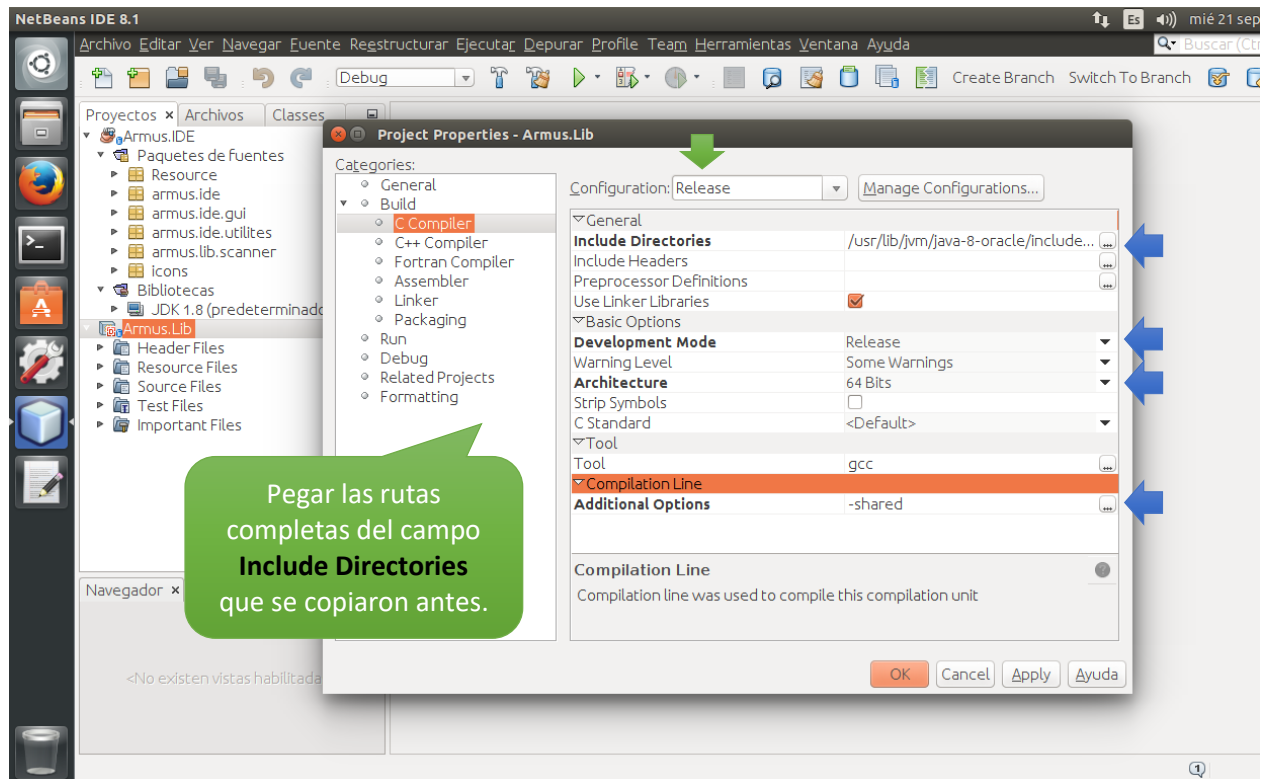
```
class Prueba {
    publica calculo3x5[] entero { //retorna un valor de tipo entero
        entero m = 0;
        m = 3 * 5;
        retornar m;
    }
    publica principal[]{
        cadena cad; //declaración de variable de tipo cadena
        entero n; //declaración de variable de tipo entero
        Sistema.mostrar["Ingrese su nombre\n"];
        Sistema.obtenerCadena[cad]; //lee desde el teclado una cadena de caracteres
        si[Sistema.comparar[cad, "carlos"]]{
            Sistema.mostrar["son iguales\n"];
        }sino{
            probar[cad]{
                caso 1: Sistema.mostrar["son diferentes\n"];
                romper;
                caso 2: cad = "nestor";
                romper;
            }
            n = Sistema.calculo3x5[];
        }
        /*comentario*/
    }
}
```

## I. Configuración para la generación de la librería de JNI para el IDE y Analizador Lexicográfico en Netbeans sobre Linux Ubuntu 14.04

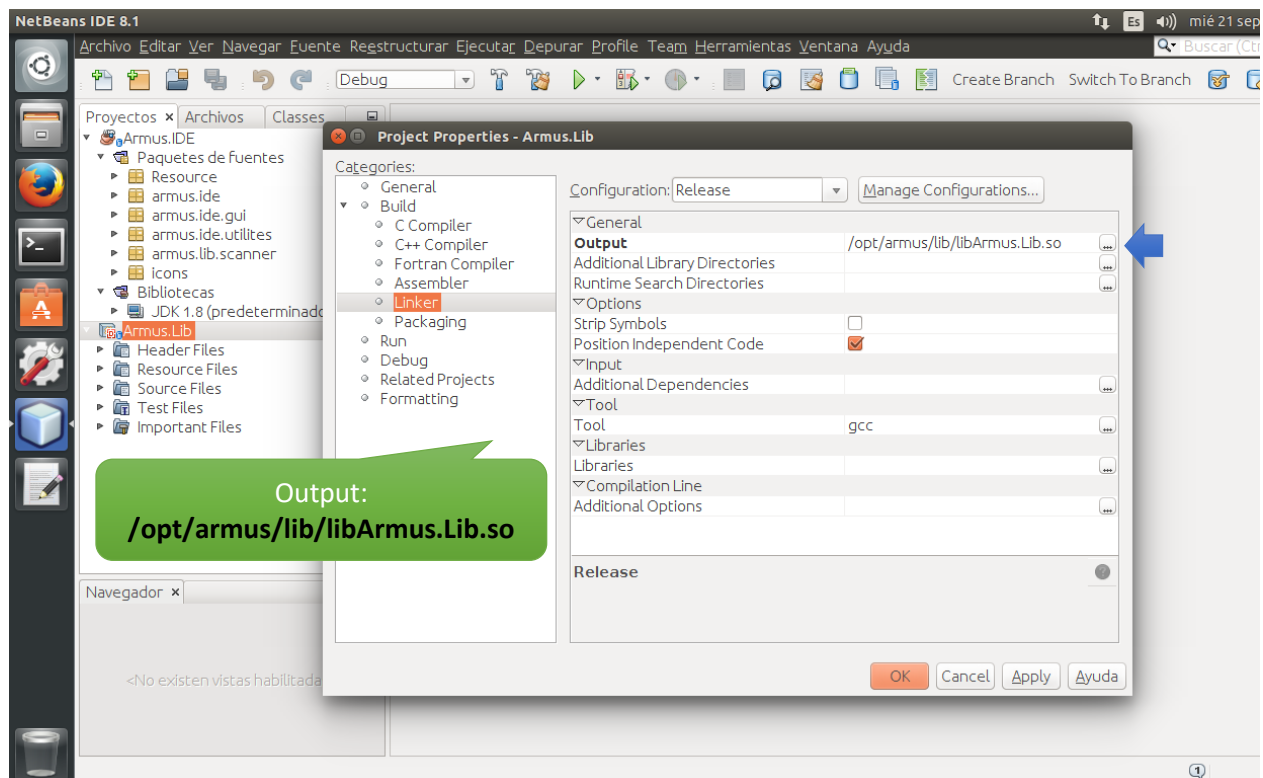
Abrimos **Netbeans** y hacemos clic derecho sobre el proyecto **Armus.Lib**, seleccionamos **Propiedades** y en la opción de **C Compiler** cambiamos en la parte de arriba Debug (active) por **Release**.



Una vez realizado ese cambio colocamos la siguiente configuración en los parámetros mostrados.



Ahora hacemos clic en la opción de **Linker** y añadimos la ruta en el campo **Output**.



Dicho directorio debe ser creado previamente con los permisos del usuario que ejecutará y generará la librería en Netbeans, para ello, abrimos una terminal (o consola) en Linux y escribimos los siguientes comandos:

<code>sudo su</code>	Accedemos como usuario root
<code>cd opt/</code>	Entramos al directorio opt
<code>mkdir armus/</code>	Creamos la carpeta armus
<code>mkdir lib/</code>	Creamos la carpeta lib

El archivo **libArmus.Lib.so** siempre debe estar en esta ruta para que el programa IDE y Analizador Léxico funcionen correctamente.

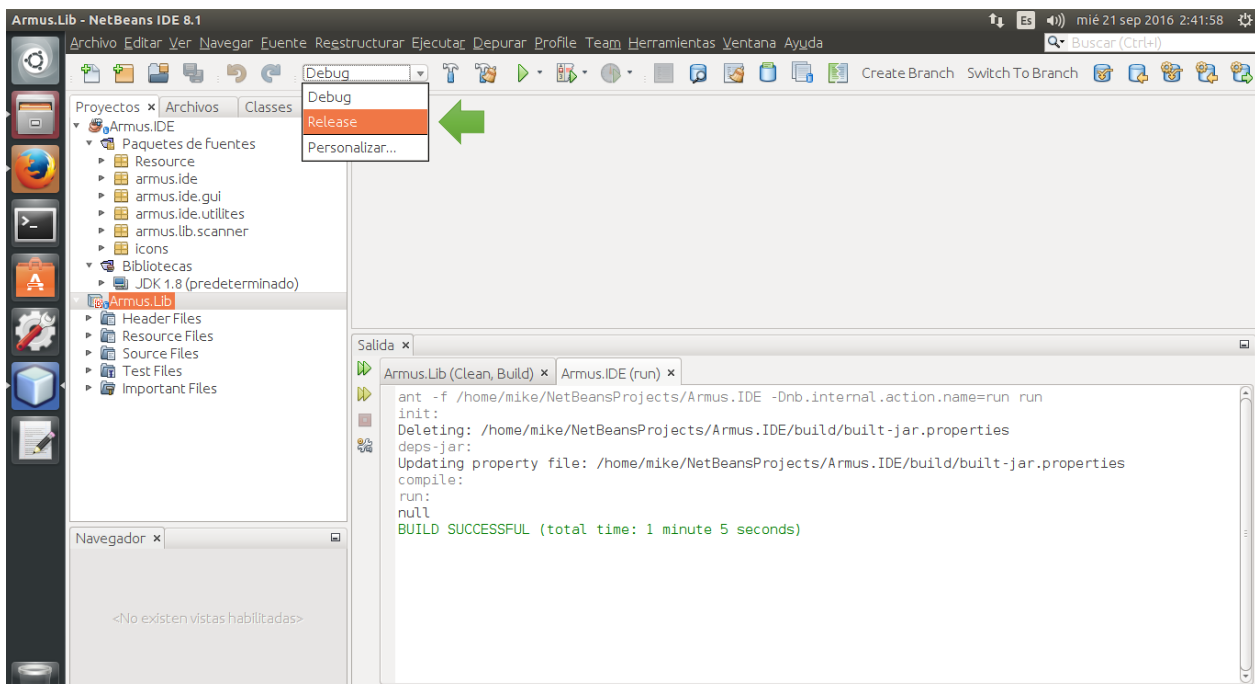
```
root@vm-ubuntu: /opt/armus/lib
root@vm-ubuntu: /opt/armus/lib#
```

Ahora creamos los permisos para el usuario **mike** en dicha carpeta:

<code>chown mike -R opt/</code>	Otorga permisos al usuario <b>mike</b> recursivamente en el directorio <b>opt</b>
---------------------------------	---

```
root@vm-ubuntu: /
root@vm-ubuntu: /opt/armus/lib# cd /
root@vm-ubuntu: /# chown mike -R opt/
```

Ahora sí, vamos nuevamente a Netbeans y en la parte superior cambiamos de Debug a Release, ejecutamos el proyecto y automáticamente la librería se creará con la configuración y ruta establecidas.



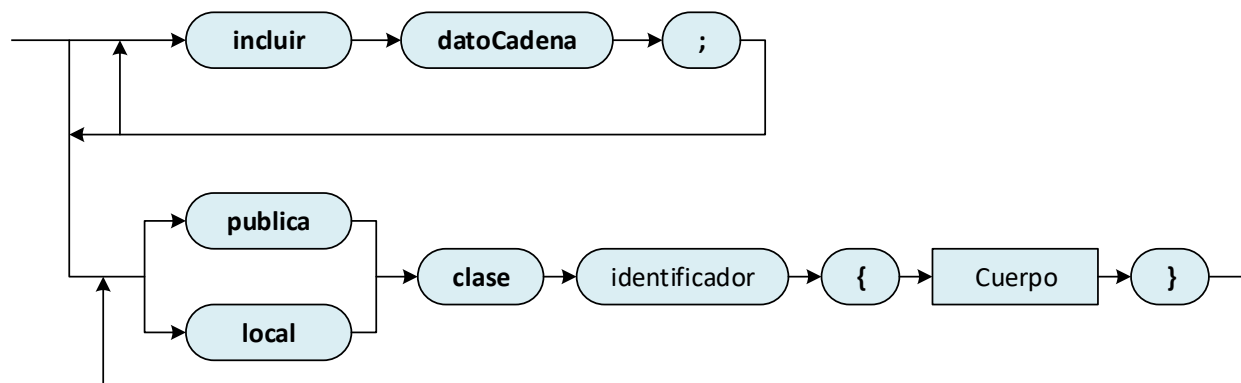
## J. Lista de Tokens utilizados

Existe una lista de los tokens utilizados, que está definida en el código fuente de la siguiente manera:

```
char *token_string[] = {"nulo", "comentario", "numeroReal", "numeroEntero", "ident", "mas", "menos",  
"por", "barra", "llavel", "llaveF", "parentI", "parentF", "corcheteI", "corcheteF", "punto", "coma",  
"puntoycoma", "asignacion", "mei", "mai", "myr", "mnr", "igl", "nig", "negacion", "ytok", "otok",  
"referencia", "enteroTok", "byteTok", "realTok", "vacioTok", "booleanoTok", "cadenaTok", "caracterTok",  
"objetoTok", "archivoTok", "siTok", "sinoTok", "probarTok", "casoTok", "defectoTok", "romperTok",  
"mientrasTok", "paraTok", "hacerTok", "paracadaTok", "sistemaTok", "obtenerEnteroTok",  
"obtenerRealTok", "obtenerCadenaTok", "obtenerCaracterTok", "mostrarTok", "publicaTok",  
"privadaTok", "retornarTok", "arregloTok", "agregarTok", "obtenerTok", "cuantosTok", "quitarTok",  
"abrirTok", "leerLineaTok", "volcadoTok", "cerrarTok", "concatenarTok", "parteEnteraTok",  
"compararTok", "mayorTok", "menorTok", "esParTok", "decimalBinTok", "potenciaTok", "absolutoTok",  
"moduloTok", "longitudCadenaTok", "claseTok", "incluirTok", "obtenerBooleanoTok", "falsoTok",  
"verdaderoTok", "datoCadena", "datoCaracter", "localTok", "principalTok", "dosPuntos"};
```

## K. Diagramas de Sintaxis

Diagrama general de un programa escrito en lenguaje Armus:

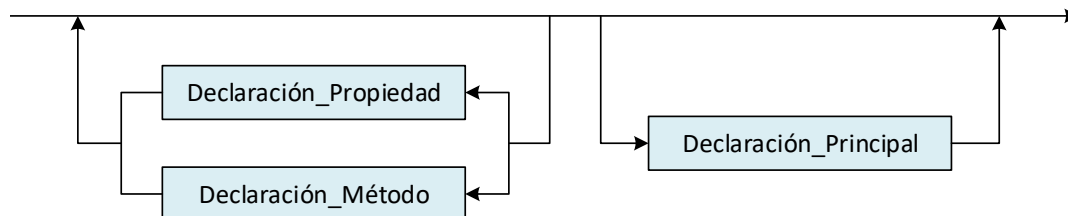


**Figura 01.** Diagrama de sintaxis de la estructura general de un algoritmo.

Las clases **públicas** son utilizadas por cualquiera, las clases **locales** solo pueden ser accedidas desde el mismo archivo

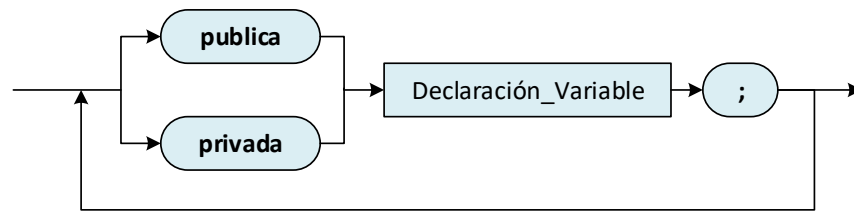
Pueden existir múltiples clases en el mismo archivo del programa

Cuerpo:



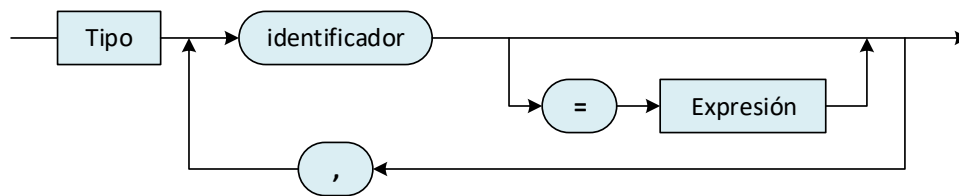
**Figura 02.** Diagrama de sintaxis para la estructura interna de una clase.

Declaración\_Propiedad:



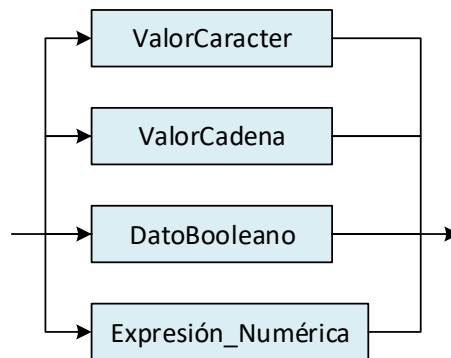
**Figura 03.** Diagrama de sintaxis para la declaración de una propiedad de la clase.

Declaración\_Variable:



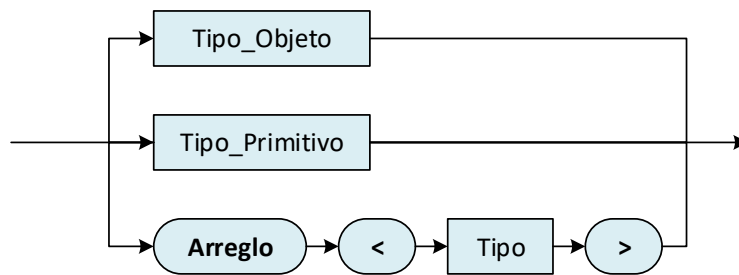
**Figura 04.** Diagrama de sintaxis para la declaración de una variable.

Expresión:



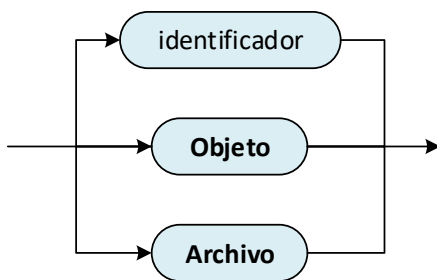
**Figura 05.** Diagrama de sintaxis que muestra las diferentes expresiones que se pueden utilizar. ValorCaracter y ValorCadena corresponden a cada una de las posibilidades que pueden generar un valor de dicho tipo dentro de una expresión.

Tipo:



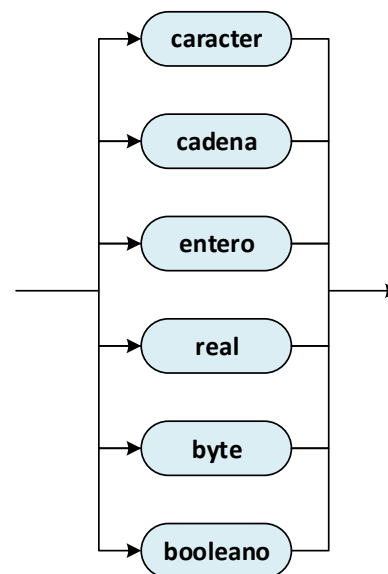
**Figura 06.** Diagrama de sintaxis que muestra las tres clasificaciones de tipo que pueden existir.

Tipo\_Objeto:



**Figura 07.** Diagrama de sintaxis que muestra los tipos objetos en el lenguaje Armus, donde identificador es una clase creada por el usuario.

Tipo\_Primitivo:



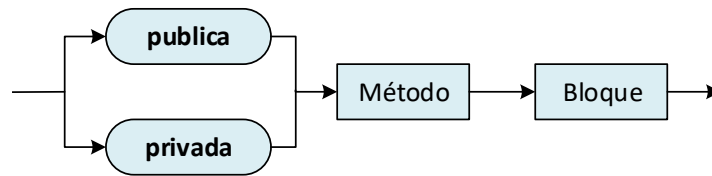
**Figura 08.** Diagrama de sintaxis que muestra los tipos primitivos (no son objetos) en el lenguaje Armus.

DatoBooleano:



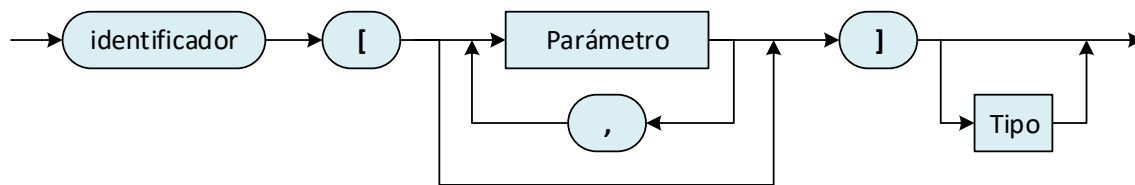
**Figura 09.** Diagrama de sintaxis para el tipo de dato booleano, solo tiene las palabras reservadas verdadero y falso.

Declaración\_Método:



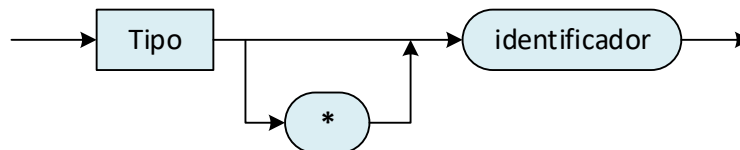
**Figura 10.** Diagrama de sintaxis que muestra la forma de declaración de un método.

Método:



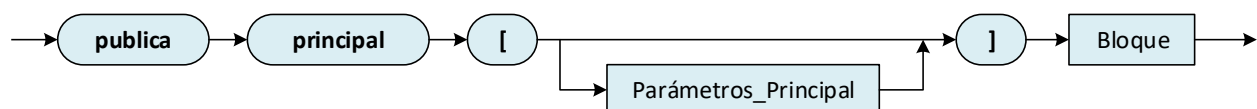
**Figura 11.** Diagrama de sintaxis que muestra la estructura sintáctica de un método, donde identificador es el nombre del método, además de que puede o no tener parámetros de entrada y si posee tipo de valor de retorno entonces es una función, sino es un procedimiento.

Parámetro:



**Figura 12.** Diagrama de sintaxis para definir un parámetro en el lenguaje Armus, el \* se utiliza opcionalmente para indicar que el parámetro será usado por referencia.

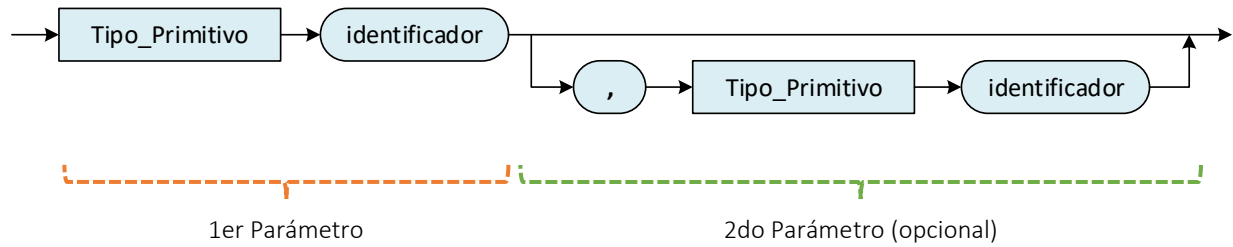
Declaración\_Principal:



**Figura 13.** Diagrama de sintaxis para definir la estructura sintáctica del método principal (main).

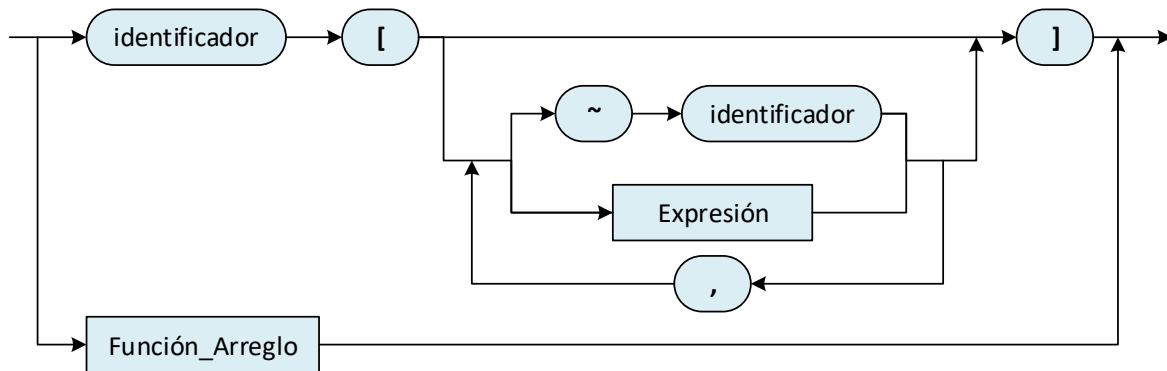


Parámetros\_Principal:



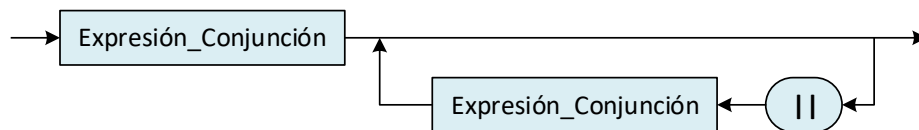
**Figura 14.** Diagrama de sintaxis de la utilización de parámetros en el método principal, tiene un máximo de dos parámetros.

Llamada\_Método:



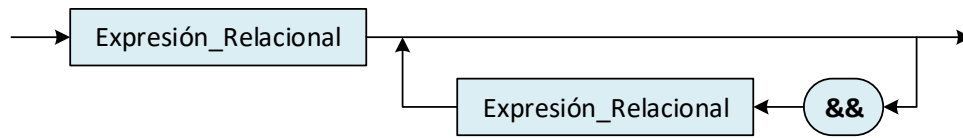
**Figura 15.** Diagrama de sintaxis de la estructura de la llamada a un método desde un objeto clase. El símbolo ~ antes de un identificador hace referencia a que este será enviado por referencia.

Expresión\_Numérica:



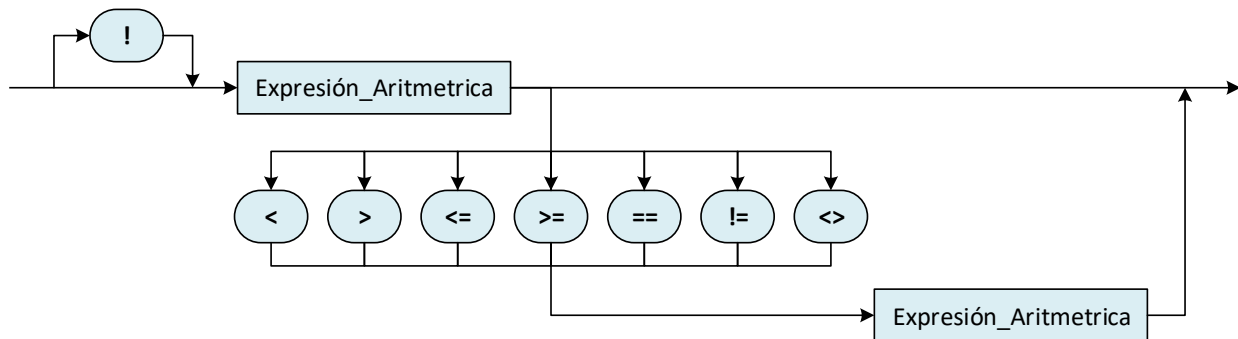
**Figura 16.** Diagrama de sintaxis para una expresión numérica, la doble barra (||) es equivalente a un OR.

Expresión\_Conjunción:



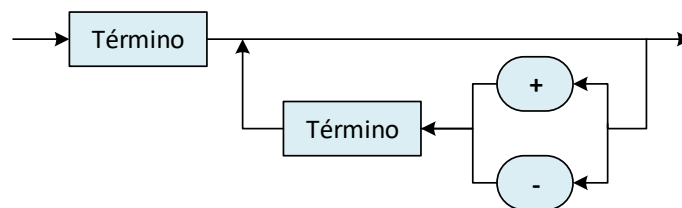
**Figura 17.** Diagrama de sintaxis para una expresión de conjunción, la doble ampersand (&&) es equivalente a un AND (lógico).

Expresión\_Relacional:



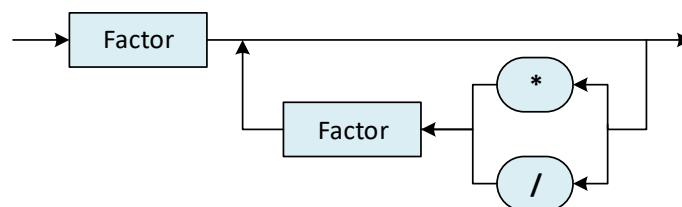
**Figura 18.** Diagrama de sintaxis para una expresión relacional, en la cual se tiene los operadores lógicos que soporta el lenguaje Armus, incluida la negación.

Expresión\_Aritmetica:



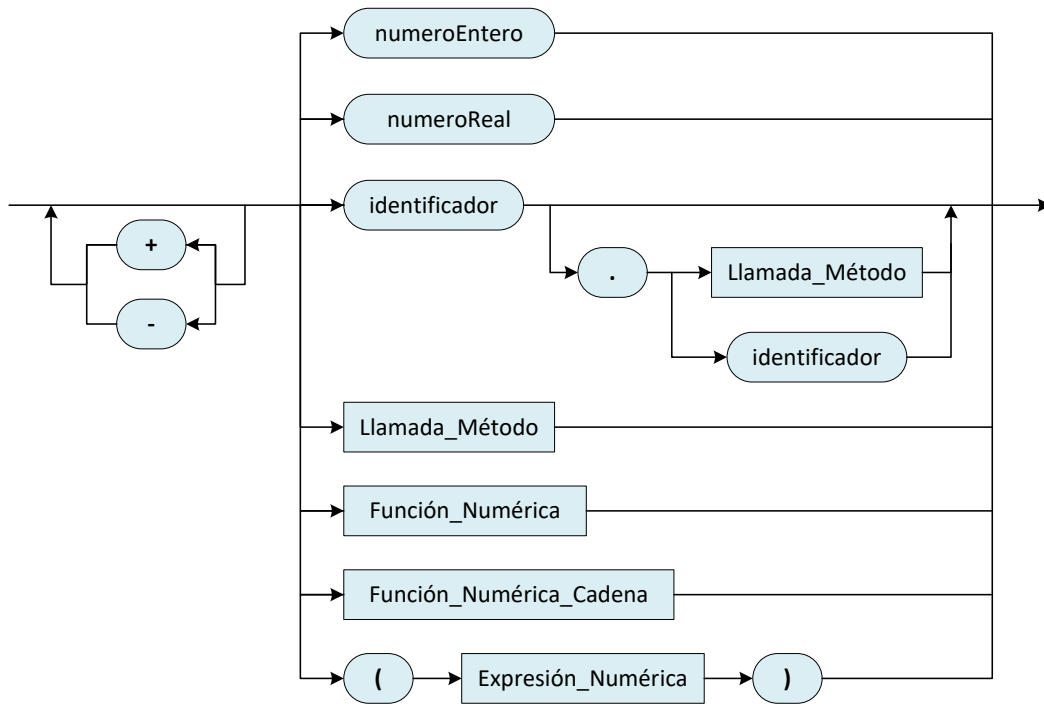
**Figura 19.** Diagrama de sintaxis de expresión aritmética, donde puede haber uno o más términos sumados o restados.

Término:



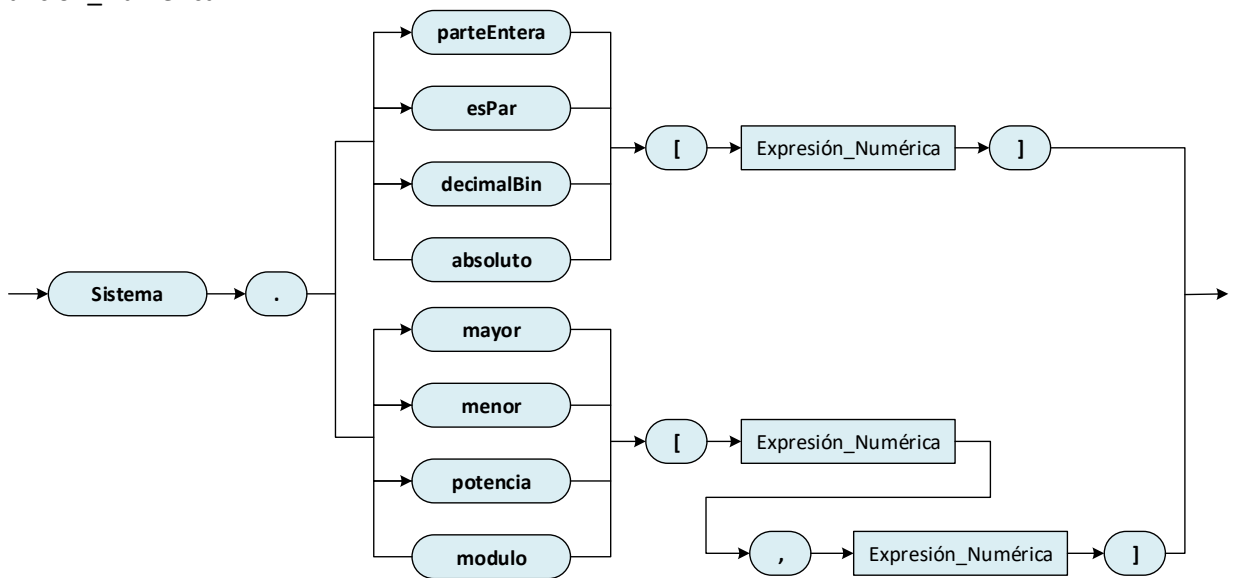
**Figura 20.** Diagrama de sintaxis de termino, la cual tiene uno o más factores multiplicados o divididos.

Factor:



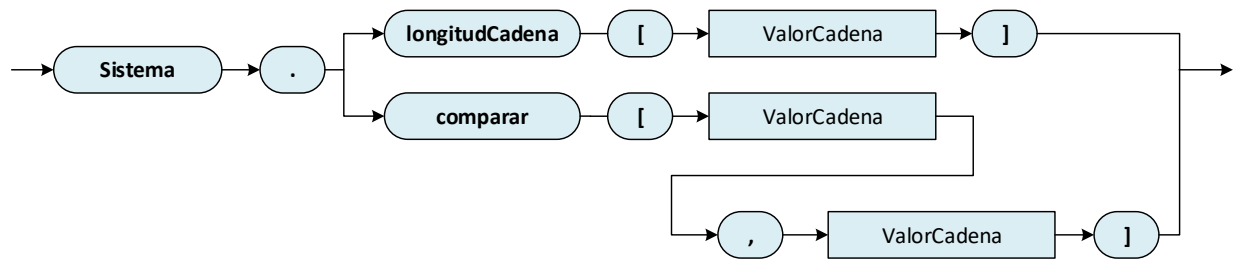
**Figura 21.** Diagrama de sintaxis de factor, el cual puede tener múltiples posibilidades, número entero, número real, variables, llamadas a métodos o atributos, funciones que devuelven valores o una expresión numérica.

Función\_Numérica:



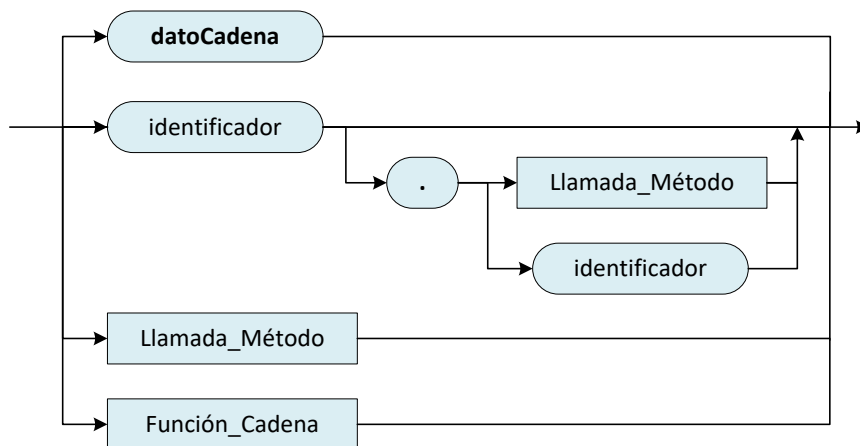
**Figura 22.** Diagrama de sintaxis para función numérica, la cual a partir del objeto Sistema se llaman las funciones predefinidas en el lenguaje.

Función\_Numérica\_Cadena:



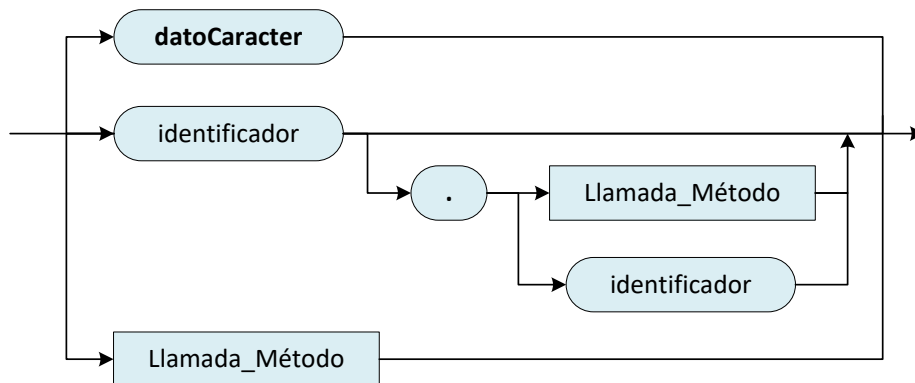
**Figura 23.** Diagrama de sintaxis para las funciones que devuelven un dato numérico recibiendo como parámetros valores de cadena.

ValorCadena:



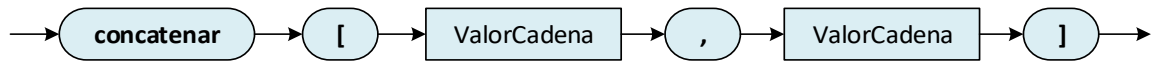
**Figura 24.** Diagrama de sintaxis para los posibles valores que pueden generar una cadena de caracteres.

ValorCaracter:



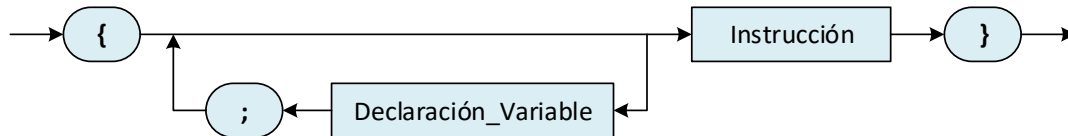
**Figura 25.** Diagrama de sintaxis para los posibles valores que puede generar un carácter.

Función\_Cadena:



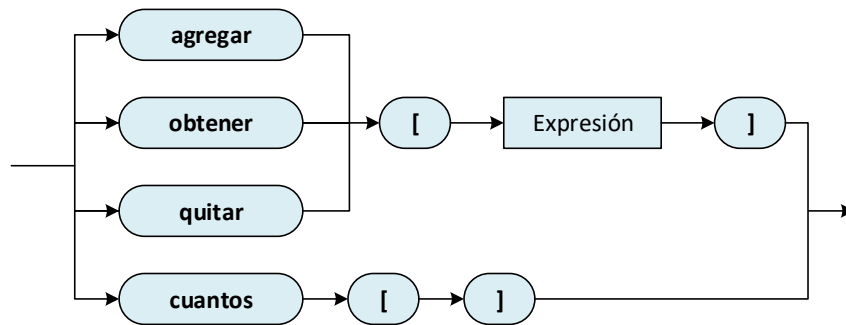
**Figura 26.** Diagrama de sintaxis de la única función que retorna una cadena de caracteres, la cual concatena dos valores de cadena en uno solo.

Bloque:



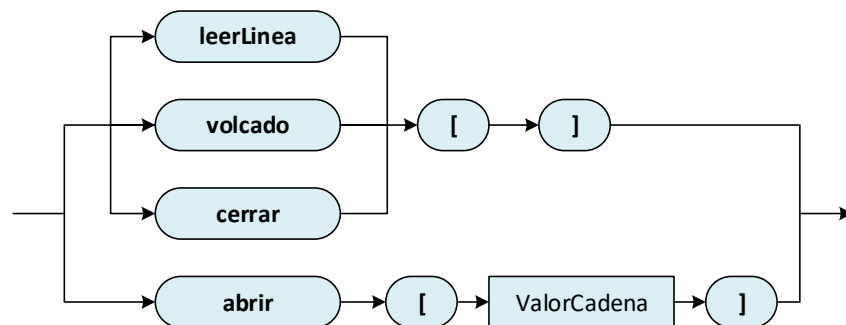
**Figura 27.** Diagrama de sintaxis de bloque, el cual declara o no variables y su contenido tiene Instrucción.

Función\_Arreglo:



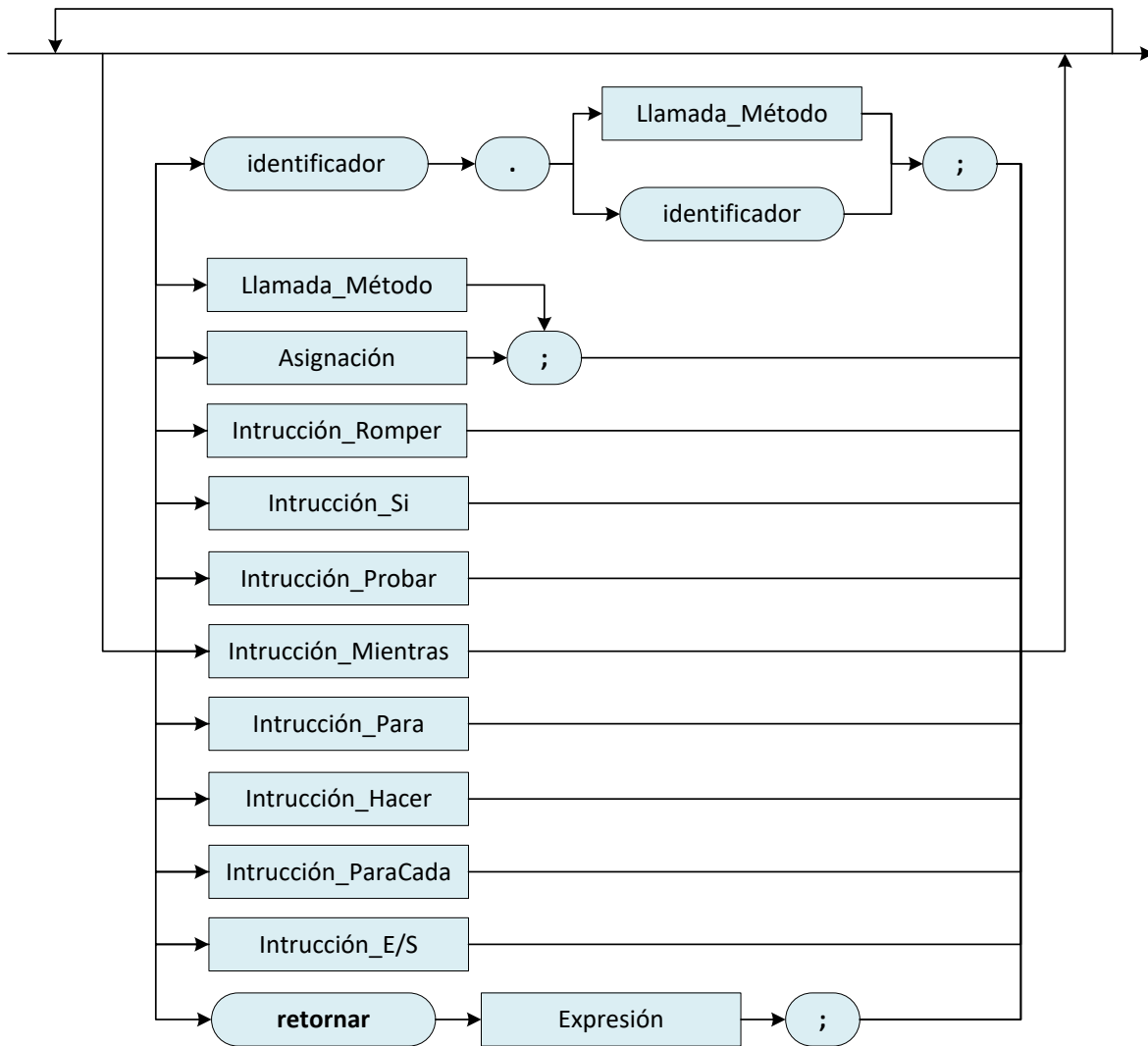
**Figura 28.** Diagrama de sintaxis de las funciones que se manejan en un arreglo, en total son cuatro.

Función\_Archivo:



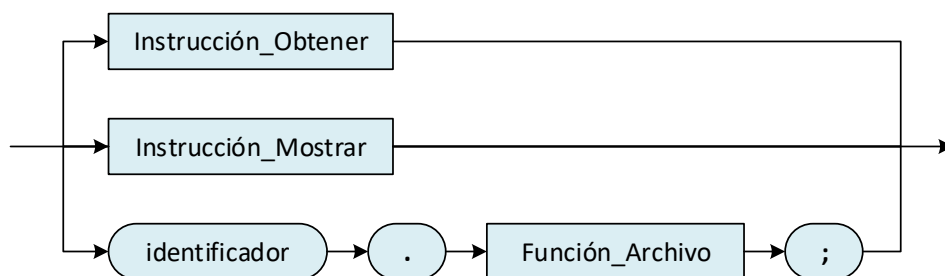
**Figura 29.** Diagrama de sintaxis de las funciones que utilizan para el manejo del objeto Archivo.

Instrucción:



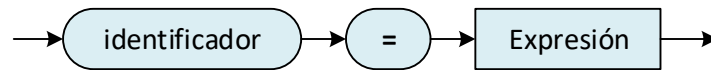
**Figura 30.** Diagrama de sintaxis para instrucción, las cuales pueden ser llamadas una o más veces.

Instrucción\_E/S:



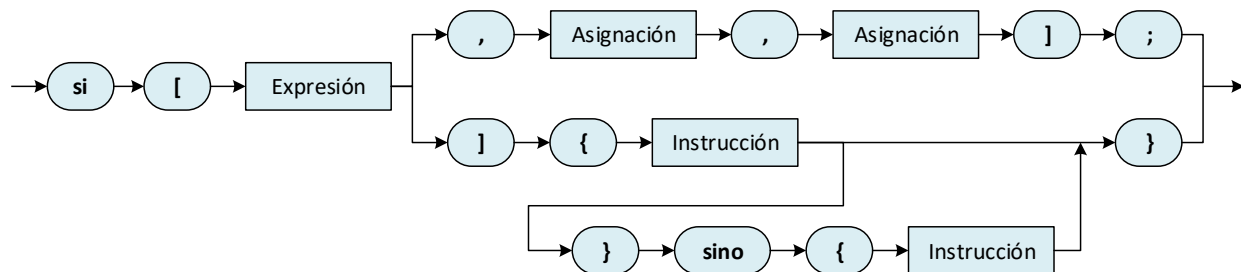
**Figura 31.** Diagrama de sintaxis para las instrucciones de entrada y salida de datos.

Asignación:



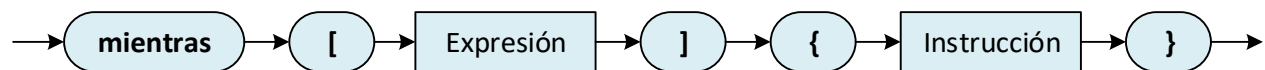
**Figura 32.** Diagrama de sintaxis que muestra la estructura de una asignación en el lenguaje.

Instrucción\_Si:



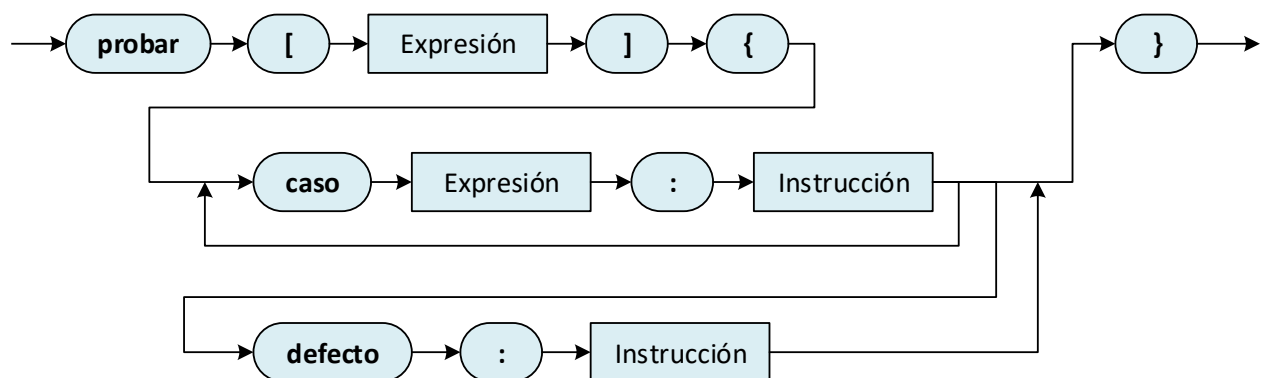
**Figura 33.** Diagrama de sintaxis para la instrucción si (if), la cual puede ser de dos formas.

Instrucción\_Mientras:



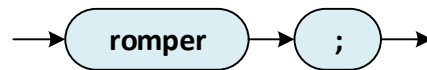
**Figura 34.** Diagrama de sintaxis para la instrucción mientras (while).

Instrucción\_Probar:



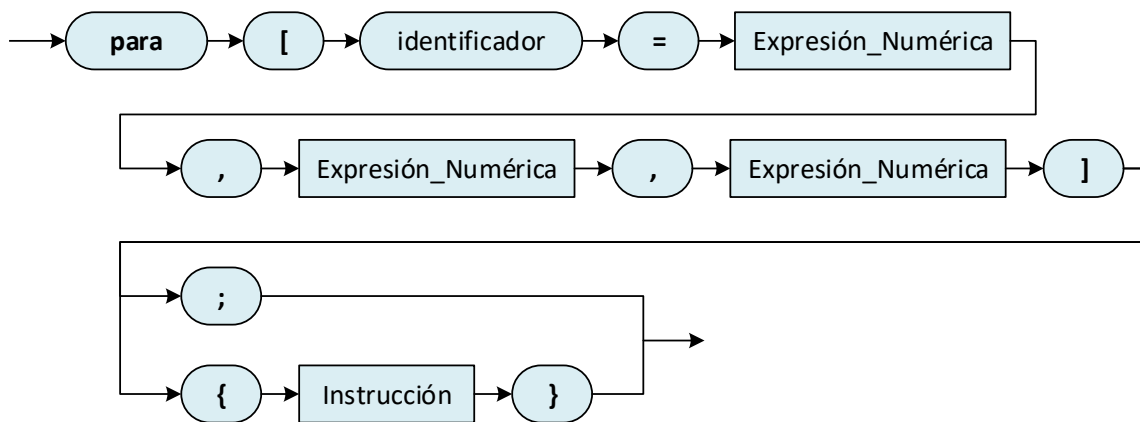
**Figura 35.** Diagrama de sintaxis para la instrucción probar (switch); tiene una opción por defecto que es opcional, pero solo puede ser utilizada una sola vez en dicha instrucción.

Instrucción\_Romper:



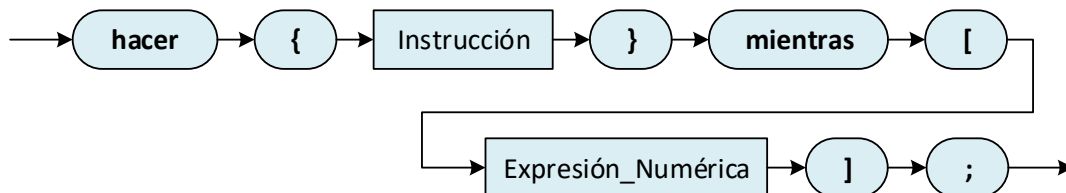
**Figura 36.** Diagrama de sintaxis para la instrucción romper, la cual simplemente debe de tener la palabra reservada romper (break), seguido de un punto y coma.

Instrucción\_Para:



**Figura 37.** Diagrama de sintaxis para la instrucción para (for), la cual puede tener instrucciones o tener un punto y coma al final.

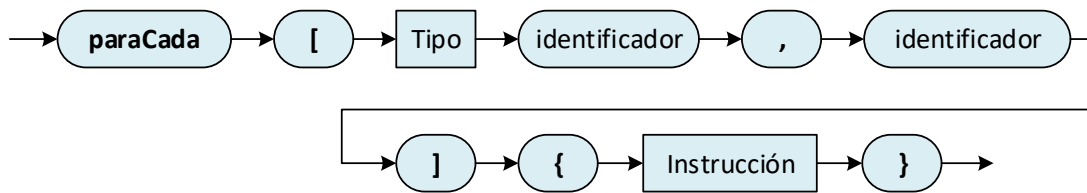
Instrucción\_Hacer:



**Figura 38.** Diagrama de sintaxis para la instrucción hacer-mientras (do-while).



Instrucción\_ParaCada:



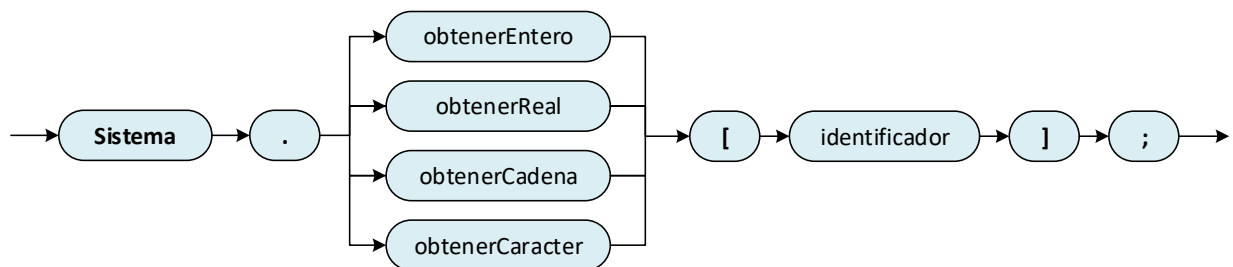
**Figura 39.** Diagrama de sintaxis para la instrucción paraCada (for-each).

Ejemplo:

```

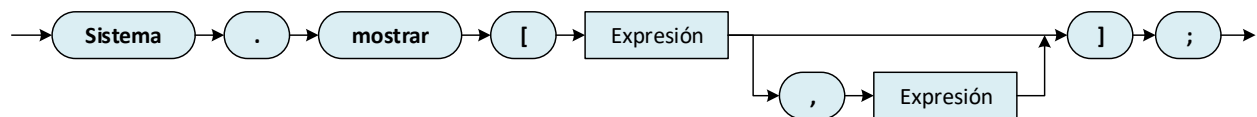
Arreglo<cadena> Lista;
Lista.agregar["lunes"];
Lista.agregar["martes"];
paraCada[cadena dia, Lista]{
    Sistema.mostrar["Día: "+dia];
}
  
```

Instrucción\_Obtener:



**Figura 40.** Diagrama de sintaxis para las instrucciones obtener del objeto sistema.

Instrucción\_Mostrar:



**Figura 41.** Diagrama de sintaxis para la instrucción mostrar.

## L. Gramática Formal

**Programa**  $\rightarrow$  **IncluirArchivo** **DefinicionClase**

**IncluirArchivo**  $\rightarrow \epsilon \mid$  incluirTok datoCadena puntoycoma **IncluirArchivo**

**DefinicionClase**  $\rightarrow$  **IdentClass** claseTok ident llaveL **Cuerpo** llaveF **MasDefinicionClase**

**IdentClass**  $\rightarrow$  publicaTok  $\mid$  localTok

**MasDefinicionClase**  $\rightarrow \epsilon \mid$  **DefinicionClase**

**Cuerpo**  $\rightarrow$  **Declar** **MasDeclar** **Declarar**

**Declar**  $\rightarrow \epsilon \mid$  **DedProp**  $\mid$  **DedMet**

**MasDeclar**  $\rightarrow \epsilon \mid$  **Declar** **MasDeclar**

**Declarar**  $\rightarrow \epsilon \mid$  **DeclararPrincipal**

**DedProp**  $\rightarrow$  **IdentProp** **Dedvar** puntoycoma

**IdentProp**  $\rightarrow$  publicaTok  $\mid$  privadaTok

**DedMet**  $\rightarrow$  **IdentProp** **Metodo** **Bloque**

**Dedvar**  $\rightarrow$  **Tipo** **Dedvariable** **MasDedvar**

**Dedvariable**  $\rightarrow$  ident **Dedvari**

**Dedvari**  $\rightarrow \epsilon \mid$  asignacion **Expresion**

**MasDedvar**  $\rightarrow \epsilon \mid$  coma **Dedvariable** **MasDedvar**

**Tipo**  $\rightarrow$  **TipoObj**  $\mid$  **TipoPrim**  $\mid$  arregloTok mnr **Tipo** myr

**TipoObj**  $\rightarrow$  ident  $\mid$  objetoTok  $\mid$  archivoTok

**TipoPrim**  $\rightarrow$  caracterTok  $\mid$  cadenaTok  $\mid$  enteroTok  $\mid$  realTok  $\mid$  byteTok  $\mid$  booleanoTok

**Expresion**  $\rightarrow$  **ValorCaracter**  $\mid$  **ValorCadena**  $\mid$  **DatoBool**  $\mid$  **ExpresNum**

**IdentLlamada**  $\rightarrow$  ident  $\mid$  ident punto **LlamadaMet**  $\mid$  ident punto ident

**DatoBool**  $\rightarrow$  verdaderoTok  $\mid$  falsoTok

**Metodo**  $\rightarrow$  ident corcheteL corcheteF **Tipo**  $\mid$  ident corcheteL **ListParam** corcheteF

**ListParam**  $\rightarrow$  **Parametro MasListParam**

**MasListParam**  $\rightarrow \epsilon$  | coma **ListParam**

**Parametro**  $\rightarrow$  **Tipo Param** ident

**Param**  $\rightarrow \epsilon$  | por

**DeclararPrincipal**  $\rightarrow$  publicaTok principalTok corcheteL **LlevaParamPrin** corcheteF **Bloque**

**LlevaParamPrin**  $\rightarrow \epsilon$  | **ParamPrin**

**ParamPrin**  $\rightarrow$  **TipoPrimIdent ParamPrinUno**

**ParamPrinUno**  $\rightarrow \epsilon$  | coma **TipoPrimIdent**

**TipoPrimIdent**  $\rightarrow$  **TipoPrim** ident

**LlamadaMet**  $\rightarrow$  **FuncionArreglo** | ident corcheteL **LlamadaMetUno** corcheteF

**LlamadaMetUno**  $\rightarrow \epsilon$  | **IdentExpres MasIdentExpres**

**IdentExpres**  $\rightarrow$  referencia ident | **Expresion**

**MasIdentExpres**  $\rightarrow \epsilon$  | coma **IdentExpres**

**ExpresNum**  $\rightarrow$  **ExpresCon MasExpresNum**

**MasExpresNum**  $\rightarrow \epsilon$  | otok **ExpresCon**

**ExpresCon**  $\rightarrow$  **ExpresRel MasExpresCon**

**MasExpresCon**  $\rightarrow \epsilon$  | ytok **ExpresRel**

**ExpresRel**  $\rightarrow$  **Negacion ExpresAri XexpresRel**

**Negacion**  $\rightarrow \epsilon$  | negacion

**XexpresRel**  $\rightarrow \epsilon$  | **Simbolo ExpresAri**

**Simbolo**  $\rightarrow$  mnr | myr | mei | mai | igl | nig

**ExpresAri**  $\rightarrow$  **Termino MasExpresAri**

**Signo**  $\rightarrow$  mas | menos

**MasExpresAri**  $\rightarrow \epsilon$  | **Signo Termino**

**Termino** → **Factor MasTermino**

**MasTermino** →  $\epsilon$  | **Oper Factor**

**Oper** → por | barra

**Factor** → **Sing Factores**

**Sing** →  $\epsilon$  | **Signo**

**Factores** → numeroEntero | numeroReal | **IdentLlamada** | **LlamadaMet** | **FuncNum** | **FuncNumCad** |  
parentI **ExpresNum** parentF

**FuncNum** → sistemaTok punto **Fnum** parentI **ExpresNum MasExpressNum** parentF

**Fnum** → **ParaNum** | **ParaSin**

**ParaNum** → parteEnteraTok | esParTok | decimalBinTok | absolutoTok

**MasExpressNum** →  $\epsilon$  | coma **ExpresNum**

**ParaSin** → mayorTok | menorTok | potenciaTok | moduloTok

**FuncNumCad** → sistemaTok punto **Conlog** corcheteI **ValorCadena MasFuncNumCad** corcheteF

**MasFuncNumCad** →  $\epsilon$  | coma **ValorCadena**

**Conlong** → compararTok | longitudCadenaTok

**ValorCaracter** → datoCaracter | **IdentLlamada** | **LlamadaMet**

**ValorCadena** → datoCadena | **IdentLlamada** | **LlamadaMet** | **FuncCad**

**FuncCad** → concatenarTok corcheteI **ValorCadena** coma **ValorCadena** corcheteF

**Bloque** → llaveI **MasBloque Instruccion** llaveF

**MasBloque** →  $\epsilon$  | **Dedvar** puntoycoma **MasBloque**

**FuncionArreglo** → **Funcarr** corcheteI **Expresion** corcheteF | cuantosTok corcheteI corcheteF

**Funcarr** → agregarTok | obtenerTok | quitarTok

**FuncionArc** → abrirTok corcheteI **ValorCadena** corcheteF | **Funcarc** corcheteI corcheteF

**Funcarc** → leerLineaTok | volcadoTok | cerrarTok

**Instruccion** →  $\epsilon$  | **MasInstruccion Instruccion**

**MasInstruccion** → **IdentLlamada** puntoycoma | **LlamadaMet** puntoycoma | **Asignacion** puntoycoma | romperTok puntoycoma | **InstSi** | **InstProbar** | **InstMientras** | **InstPara** | **InstHacer** | **InstParaCad** | **InstES** | retornarTok **Expresion** puntoycoma

**InstES** → **InstObtener** | **InstMostrar** | ident punto **FuncionArc** puntoycoma

**Asignacion** → ident asignacion **Expresion**

**InstSi** → siTok corcheteI **Expresion** **InstSiUno**

**InstSiUno** → coma **Asignacion** coma **Asignacion** corcheteF puntoycoma | corcheteF llaveI **Instruccion** **InstSiDos** llaveF

**InstSiDos** → ε | llaveF sinoTok llaveI **Instruccion**

**InstMientras** → mientrasTok corcheteI **Expresion** corcheteF llaveI **Instruccion** llaveF

**InstProbar** → probarTok corcheteI **Expresion** corcheteF llaveI casoTok **Expresion** dospuntos **Instruccion** **MasInstProbar** **InstProbarUno** llaveF

**InstProbarUno** → ε | defectoTok dospuntos **Instruccion**

**MasInstProbar** → ε | casoTok **Expresion** dospuntos **Instruccion**

**InstPara** → paraTok corcheteI ident asignacion **ExpresNum** coma **ExpresNum** coma **ExpresNum** corcheteF **InstParaUno**

**InstParaUno** → puntoycoma | llaveI **Instruccion** llaveF

**InstHacer** → hacerTok llaveI **Instruccion** llaveF mientrasTok corcheteI **ExpresNum** corcheteF puntoycoma

**InstParaCad** → paracadaTok corcheteI **Tipo** ident coma ident corcheteF llaveI **Instruccion** llaveF

**InstObtener** → sistemaTok punto **Obtener** corcheteI ident corcheteF puntoycoma

**Obtener** → obtenerEnteroTok | obtenerRealTok | obtenerCadenaTok | obtenerCaracterTok

**InstMostrar** → sistemaTok punto mostrarTok corcheteI **Expresion** **InstMostrarUno** corcheteF puntoycoma

**InstMostrarUno** → ε | coma **Expresion**

## M. Aplicación de la Regla de Primero

Prim(**Programa**) = {incluirTok} U {}

Prim(**IncluirArchivo**) = {incluirTok} U {}

Prim(**DefinicionClase**) = {publicaTok, localTok} U {}

Prim(**IdentClass**) = {publicaTok, localTok} U {}

Prim(**MasDefinicionClase**) = {publicaTok, localTok} U {}

Prim(**Cuerpo**) = Prim(Declar) U Prim(MasDeclar) U Prim(Declarar) = {publicaTok, privadaTok} U {}

Prim(**Declar**) = Prim(DedProp) U Prim(DedMet) U {} = {publicaTok, privadaTok} U {}

Prim(**MasDeclar**) = {publicaTok, privadaTok} U {}

Prim(**Declarar**) = Prim(DedPrim) U {} = {publicaTok} U {}

Prim(**DedProp**) = {publicaTok, privadaTok}

Prim(**IdentProp**) = {publicaTok, privadaTok}

Prim(**DetMet**) = {publicaTok, privadaTok}

Prim(**Dedvar**) = {ident, objetoTok, archivoTok, caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok, arregloTok}

Prim(**Dedvariable**) = {ident}

Prim(**Dedvari**) = {asignacion} U {}

Prim(**MasDedvar**) = {coma} U {}

Prim(**Tipo**) = Prim(TipoObj) U Prim(TipoPrim) U {arregloTok} = {ident, objetoTok, archivoTok, caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok, arregloTok}

Prim(**TipoObj**) = {ident, objetoTok, archivoTok}

Prim(**TipoPrim**) = {caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok}

Prim(**Expresion**) = Prim(ValorCaracter) U Prim(ValorCadena) U Prim(DatoBool) U Prim(ExpresNum) = {datoCaracter, ident, agregarTok, obtenerTok, quitarTok, cuantosTok, datoCadena, concatenarTok, verdaderoTok, falsoTok, negacion, mas, menos, numeroEntero, numeroReal, parentI, sistemaTok} U {}

Prim(**IdentLlamada**) = {ident}

Prim(**DatoBool**) = {verdaderoTok, falsoTok}

Prim(**Metodo**) = {ident}

Prim(**ListParam**) = {ident, objetoTok, archivoTok, caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok, arregloTok}

Prim(**MasListParam**) = {coma} U {}

Prim(**Parametro**) = {ident, objetoTok, archivoTok, caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok, arregloTok}

Prim(**Param**) = {por} U {}

Prim(**DeclararPrincipal**) = {publicaTok}

Prim(**LlevaParamPrin**) = Prim(ParamPrin) U {} = {caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok} U {}

Prim(**ParamPrin**) = Prim(TipoPrimIdent) = Prim(TipoPrim) = {caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok}

Prim(**ParamPrinUno**) = {coma} U {}

Prim(**TipoPrimIdent**) = Prim(TipoPrim) = {caracterTok, cadenaTok, enteroTok, realTok, byteTok, booleanoTok}

Prim(**LLamadaMet**) = Prim(FuncionArreglo) U {ident} = {ident, agregarTok, obtenerTok, quitarTok, cuantosTok}

Prim(**LlamadaMetUno**) = Prim(IdentExpres) U {} = {referencia, datoCaracter, ident, datoCadena, concatenarTok, verdaderoTok, falsoTok, negación} U {}

Prim(**IdentExpres**) = {referencia} U Prim(Expresion) = {referencia, datoCaracter, ident, datoCadena, concatenarTok, verdaderoTok, falsoTok, negacion} U {}

Prim(**MasIdentExpres**) = {coma} U {}

Prim(**ExpresNum**) = Prim(ExpresCon) = Prim(ExpresRel) = Prim(Negacion) = {negacion} U Prim(ExpresAri)  
= {negacion} U Prim(Termino)  
= {negacion} U Prim(Factor)  
= {negacion, mas, menos, numeroEntero, numeroReal, parentl, ident} U Prim(LlamadaMet) U  
Prim(FuncNum) U Prim(FuncNumCad)  
= {negacion, mas, menos, numeroEntero, numeroReal, parentl, ident, agregarTok, obtenerTok,  
quitarTok, cuantosTok, sistemaTok}

Prim(**MasExpresNum**) = {oTok} U {}

$\text{Prim}(\mathbf{ExpresCon}) = \text{Prim}(\text{ExpresRel}) = \text{Prim}(\text{Negacion}) = \{\text{negacion}, \text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \cup \{\}$

$\text{Prim}(\mathbf{MasExpresCon}) = \{\text{yTok}\} \cup \{\}$

$\text{Prim}(\mathbf{ExpresRel}) = \text{Prim}(\text{Negacion}) = \{\text{negacion}, \text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \cup \{\}$

$\text{Prim}(\mathbf{Negacion}) = \{\text{negacion}\} \cup \{\}$

$\text{Prim}(\mathbf{XexpresRel}) = \text{Prim}(\text{Simbolo}) \cup \{\} = \{\text{mnr}, \text{myr}, \text{mei}, \text{mai}, \text{igl}, \text{nig}\} \cup \{\}$

$\text{Prim}(\mathbf{Simbolo}) = \{\text{mnr}, \text{myr}, \text{mei}, \text{mai}, \text{igl}, \text{nig}\}$

$\text{Prim}(\mathbf{ExpresAri}) = \text{Prim}(\text{Termino}) = \text{Prim}(\text{Factor}) = \text{Prim}(\text{Sing}) = \text{Prim}(\text{Signo}) \cup \{\} = \{\text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \cup \{\}$

$\text{Prim}(\mathbf{Signo}) = \{\text{mas}, \text{menos}\}$

$\text{Prim}(\mathbf{MasExpresAri}) = \text{Prim}(\text{Signo}) \cup \{\} = \{\text{mas}, \text{menos}\} \cup \{\}$

$\text{Prim}(\mathbf{Termino}) = \text{Prim}(\text{Factor}) \cup \text{Prim}(\text{Sing}) = \text{Prim}(\text{Factor}) \cup \text{Prim}(\text{Signo}) \cup \{\} = \{\text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \cup \{\}$

$\text{Prim}(\mathbf{MasTermino}) = \text{Prim}(\text{Oper}) \cup \{\} = \{\text{por}, \text{barra}\} \cup \{\}$

$\text{Prim}(\mathbf{Oper}) = \{\text{por}, \text{barra}\}$

$\begin{aligned} \text{Prim}(\mathbf{Factor}) &= \text{Prim}(\text{Sing}) \cup \text{Prim}(\text{Factores}) \cup \{\} \\ &= \text{Prim}(\text{Signo}) \cup \text{Prim}(\text{Factores}) \cup \{\} \\ &= \{\text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \cup \{\} \end{aligned}$

$\text{Prim}(\mathbf{Sing}) = \text{Prim}(\text{Signo}) \cup \{\} = \{\text{mas}, \text{menos}\} \cup \{\}$

$\begin{aligned} \text{Prim}(\mathbf{Factores}) &= \{\text{numeroEntero}, \text{numeroReal}, \text{parentl}\} \cup \text{Prim}(\text{IdentLlamada}) \cup \text{Prim}(\text{LlamadaMet}) \cup \\ &\text{Prim}(\text{FuncNum}) \cup \text{Prim}(\text{FuncNumCad}) = \{\text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\} \end{aligned}$

$\text{Prim}(\mathbf{FuncNum}) = \{\text{sistemaTok}\}$

$\text{Prim}(\mathbf{Fnum}) = \text{Prim}(\text{ParaNum}) \cup \text{Prim}(\text{ParaSin}) = \{\text{parteEnteraTok}, \text{esParTok}, \text{decimalBinTok}, \text{absolutoTok}, \text{mayorTok}, \text{menorTok}, \text{potenciaTok}, \text{moduloTok}\}$

$\text{Prim}(\mathbf{ParaNum}) = \{\text{parteEnteraTok}, \text{esParTok}, \text{decimalBinTok}, \text{absolutoTok}\}$



$\text{Prim}(\text{MasExpressNum}) = \{\text{coma}\} \cup \{\}$   
 $\text{Prim}(\text{ParaSin}) = \{\text{mayorTok}, \text{menorTok}, \text{potenciaTok}, \text{moduloTok}\}$   
 $\text{Prim}(\text{FuncNumCad}) = \{\text{sistemaTok}\}$   
 $\text{Prim}(\text{MasFuncNumCad}) = \{\text{coma}\} \cup \{\}$   
 $\text{Prim}(\text{Conlong}) = \{\text{compararTok}, \text{longitudCadenaTok}\}$   
 $\text{Prim}(\text{ValorCaracter}) = \{\text{datoCaracter}\} \cup \text{Prim}(\text{IdentLlamada}) \cup \text{Prim}(\text{LLamadaMet}) = \{\text{datoCaracter}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}\}$   
 $\text{Prim}(\text{ValorCadena}) = \{\text{datoCadena}\} \cup \text{Prim}(\text{IdentLlamada}) \cup \text{Prim}(\text{LLamadaMet}) \cup \text{Prim}(\text{FuncCad}) = \{\text{datoCadena}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{concatenarTok}\}$   
 $\text{Prim}(\text{FuncCad}) = \{\text{concatenarTok}\}$   
 $\text{Prim}(\text{Bloque}) = \{\text{llavel}\}$   
 $\text{Prim}(\text{MasBloque}) = \text{Prim}(\text{Dedvar}) \cup \{\} = \{\text{ident}, \text{objetoTok}, \text{archivoTok}, \text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{booleanoTok}, \text{arregloTok}\} \cup \{\}$   
 $\text{Prim}(\text{FuncionArreglo}) = \text{Prim}(\text{Funcarr}) \cup \{\text{cuantosTok}\} = \{\text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}\}$   
 $\text{Prim}(\text{Funcarr}) = \{\text{agregarTok}, \text{obtenerTok}, \text{quitarTok}\}$   
 $\text{Prim}(\text{FuncionArc}) = \{\text{abrirTok}\} \cup \text{Prim}(\text{Funcarc}) = \{\text{abrirTok}, \text{leerLineaTok}, \text{volcadoTok}, \text{cerrarTok}\}$   
 $\text{Prim}(\text{Funcarc}) = \{\text{leerLineaTok}, \text{volcadoTok}, \text{cerrarTok}\}$   
 $\text{Prim}(\text{Instruccion}) = \text{Prim}(\text{MasInstruccion}) \cup \{\} = \{\text{romperTok}, \text{retornarTok}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{siTok}, \text{probarTok}, \text{mientrasTok}, \text{paraTok}, \text{hacerTok}, \text{paracadaTok}, \text{sistemaTok}\} \cup \{\}$   
 $\text{Prim}(\text{MasInstruccion}) = \{\text{romperTok}, \text{retornarTok}\} \cup \text{Prim}(\text{IdentLlamada}) \cup \text{Prim}(\text{LLamadaMet}) \cup \text{Prim}(\text{Asignacion}) \cup \text{Prim}(\text{InstSi}) \cup \text{Prim}(\text{InstProbar}) \cup \text{Prim}(\text{InstMientras}) \cup \text{Prim}(\text{InstPara}) \cup \text{Prim}(\text{InstHacer}) \cup \text{Prim}(\text{InstParaCad}) \cup \text{Prim}(\text{InstES}) = \{\text{romperTok}, \text{retornarTok}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{siTok}, \text{probarTok}, \text{mientrasTok}, \text{paraTok}, \text{hacerTok}, \text{paracadaTok}, \text{sistemaTok}\}$   
 $\text{Prim}(\text{InstES}) = \text{Prim}(\text{InstObtener}) \cup \text{Prim}(\text{InstMostrar}) \cup \{\text{ident}\} = \{\text{sistemaTok}, \text{ident}\}$   
 $\text{Prim}(\text{Asignacion}) = \{\text{ident}\}$   
 $\text{Prim}(\text{InstSi}) = \{\text{siTok}\}$   
 $\text{Prim}(\text{InstSiUno}) = \{\text{coma}, \text{corcheteF}\}$

$\text{Prim}(\text{InstSiDos}) = \{\text{llaveF}\} \cup \{\}$

$\text{Prim}(\text{InstMientras}) = \{\text{mientrasTok}\}$

$\text{Prim}(\text{InstProbar}) = \{\text{probarTok}\}$

$\text{Prim}(\text{InstProbarUno}) = \{\text{defectoTok}\} \cup \{\}$

$\text{Prim}(\text{MasInstProbar}) = \{\text{casoTok}\} \cup \{\}$

$\text{Prim}(\text{InstPara}) = \{\text{paraTok}\}$

$\text{Prim}(\text{InstParaUno}) = \{\text{llavel}, \text{puntoycoma}\}$

$\text{Prim}(\text{InstHacer}) = \{\text{hacerTok}\}$

$\text{Prim}(\text{InstParaCad}) = \{\text{paracadaTok}\}$

$\text{Prim}(\text{InstObtener}) = \{\text{sistemaTok}\}$

$\text{Prim}(\text{Obtener}) = \{\text{obtenerEnteroTok}, \text{obtenerRealTok}, \text{obtenerCadenaTok}, \text{obtenerCaracterTok}\}$

$\text{Prim}(\text{InstMostrar}) = \{\text{sistemaTok}\}$

$\text{Prim}(\text{InstMostrarUno}) = \{\text{coma}\} \cup \{\}$

## N. Aplicación de las Reglas de Primero y Siguiente

### Definición de gramática LL(1)

Para que una gramática independiente de contexto cumpla con ser una gramática LL1 se debe la siguiente regla:

Dada una Gramática libre de contexto G(CFG) y sus producciones de la forma:

$\zeta \rightarrow \zeta_1 \mid \zeta_2 \mid \dots \mid \zeta_n$  se debe cumplir que:

$$\text{Prim}(\zeta_1) \cap \text{Prim}(\zeta_2) \cap \dots \cap \text{Prim}(\zeta_n) = \emptyset$$

Luego, si se puede derivar a  $\epsilon$  (palabra vacía) de un símbolo no terminal X, se requiere además que:

$$\text{Prim}(X) \cap \text{Sig}(X) = \emptyset$$

A continuación, se presenta las evaluaciones matemáticas de las reglas de Primero y Siguiente, utilizadas para comprobar si la gramática que genera al Lenguaje **Armus** es LL(1):

\*\*\*\*\*

**Programa**  $\rightarrow$  **IncluirArchivo** **DefinicionClase**

**IncluirArchivo**  $\rightarrow \epsilon \mid$  incluirTok datoCadena puntoycoma **IncluirArchivo**

#### Regla 01

$\text{Prim}(\text{IncluirArchivo1}) \cap \text{Prim}(\text{IncluirArchivo2}) = \{\} \cap \{\text{incluirTok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{IncluirArchivo}) \cap \text{Sig}(\text{IncluirArchivo}) = \text{Prim}(\text{IncluirArchivo}) \cap \text{Prim}(\text{DefinicionClase}) = \{\text{incluirTok}\} \cap \{\text{publicaTok}, \text{localTok}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción "**IncluirArchivo**".

\*\*\*\*\*

**DefinicionClase**  $\rightarrow$  **IdentClass** claseTok ident llavel **Cuerpo** llaveF **MasDefinicionClase**

**IdentClass**  $\rightarrow$  publicaTok  $\mid$  localTok

**MasDefinicionClase**  $\rightarrow \epsilon \mid$  **DefinicionClase**

#### Regla 01

$\text{Prim}(\text{MasDefinicionClase1}) \cap \text{Prim}(\text{MasDefinicionClase2}) = \{\} \cap \{\text{publicaTok}, \text{localTok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasDefinicionClase}) \cap \text{Sig}(\text{MasDefinicionClase}) = \{\text{publicaTok}, \text{localTok}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción "**MasDefinicionClase**".

\*\*\*\*\*

**Cuerpo**  $\rightarrow$  **Declar** **MasDeclar** **Declarar**

**Declar**  $\rightarrow \epsilon \mid$  **DedProp**  $\mid$  **DedMet**

**DedProp**  $\rightarrow$  **IdentProp** **Dedvar** puntoycoma

**DedMet** → **IdentProp Metodo Bloque**

**IdentProp** → publicaTok | privadaTok

Regla 01

$\text{Prim}(\text{Declar1}) \cap \text{Prim}(\text{Declar2}) \cap \text{Prim}(\text{Declar3})$

$= \{\} \cap \text{Prim}(\text{DedProp}) \cap \text{Prim}(\text{DedMet})$

$= \{\} \cap \text{Prim}(\text{IdentProp}) \cap \text{Prim}(\text{IdentProp})$

$= \{\} \cap \{\text{publicaTok}, \text{privadaTok}\} \cap \{\text{publicaTok}, \text{privadaTok}\}$

$= \{\text{publicaTok}, \text{privadaTok}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**Declar**”.

Semántica: Se lee el siguiente token para identificar si es un método o atributo (propiedad)

\*\*\*\*\*

**Cuerpo** → **Declar MasDeclar Declarar**

**Declar** → ε | **DedProp** | **DedMet**

**MasDeclar** → ε | **Declar MasDeclar**

**DedProp** → **IdentProp Dedvar** puntoycoma

**DedMet** → **IdentProp Metodo Bloque**

**IdentProp** → publicaTok | privadaTok

**Declarar** → ε | **DeclararPrincipal**

**DeclararPrincipal** → publicaTok principalTok corcheteL **LlevaParamPrin** corcheteF **Bloque**

Regla 01

$\text{Prim}(\text{MasDeclar1}) \cap \text{Prim}(\text{MasDeclar2})$

$= \{\} \cap \text{Prim}(\text{Declar})$

$= \{\} \cap \{\text{publicaTok}, \text{privadaTok}\} \cap \{\}$

$= \emptyset$

Regla 02

$\text{Prim}(\text{MasDeclar}) \cap \text{Sig}(\text{MasDeclar})$

$= \{\text{publicaTok}, \text{privadaTok}\} \cap \text{Prim}(\text{Declarar})$

$= \{\text{publicaTok}, \text{privadaTok}\} \cap \{\text{publicaTok}\} \cup \{\}$

$= \{\text{publicaTok}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**MasDeclar**”.

Semántica: Se lee el siguiente token para identificar si es un método o atributo (propiedad)

\*\*\*\*\*

**Cuerpo** → **Declar MasDeclar Declarar**

**Declarar** → ε | **DeclararPrincipal**

**DeclararPrincipal** → publicaTok principalTok corcheteL **LlevaParamPrin** corcheteF **Bloque**

Regla 01

$\text{Prim}(\text{Declarar1}) \cap \text{Prim}(\text{Declarar2})$

$= \{\} \cap \text{Prim}(\text{DeclararPrincipal})$

$= \{\} \cap \{\text{publicaTok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{Declarar}) \cap \text{Sig}(\text{Declarar}) = \{\text{publicaTok}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**Declarar**”.

\*\*\*\*\*

**Dedvariable** → ident **Dedvari**

**Dedvari** → ε | asignacion **Expresion**

#### Regla 01

$\text{Prim}(\text{Dedvari1}) \cap \text{Prim}(\text{Dedvar2}) = \{\} \cap \{\text{asignacion}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{Dedvari}) \cap \text{Sig}(\text{Dedvari}) = \{\text{asignacion}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**Dedvari**”.

\*\*\*\*\*

**Dedvar** → **Tipo** **Dedvariable** **MasDedvar**

**MasDedvar** → ε | coma **Dedvariable** **MasDedvar**

#### Regla 01

$\text{Prim}(\text{MasDedvar1}) \cap \text{Prim}(\text{MasDedvar2}) = \{\} \cap \{\text{coma}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasDedvar}) \cap \text{Sig}(\text{MasDedvar}) = \{\text{coma}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasDedvar**”.

\*\*\*\*\*

**Metodo** → ident corcheteI corcheteF **Tipo** | ident corcheteI **ListParam** corcheteF

**ListParam** → **Parametro** **MasListParam**

**MasListParam** → ε | coma **ListParam**

#### Regla 01

$\text{Prim}(\text{MasListParam1}) \cap \text{Prim}(\text{MasListParam2}) = \{\} \cap \{\text{coma}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasListParam}) \cap \text{Sig}(\text{MasListParam}) = \{\text{coma}\} \cap \{\text{corcheteF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasListParam**”.

\*\*\*\*\*

**Parametro** → **Tipo** **Param** ident

**Param** → ε | por

#### Regla 01

$\text{Prim}(\text{Param1}) \cap \text{Prim}(\text{Param2}) = \{\} \cap \{\text{por}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{Param}) \cap \text{Sig}(\text{Param}) = \{\text{por}\} \cap \{\text{ident}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “Param”.

\*\*\*\*\*

**DeclararPrincipal** → publicaTok principalTok corcheteL **LlevaParamPrin** corcheteF **Bloque**

**LlevaParamPrin** → ε | **ParamPrin**

**ParamPrin** → **TipoPrimIdent** **ParamPrinUno**

**TipoPrimIdent** → **TipoPrim** ident

**TipoPrim** → caracterTok | cadenaTok | enteroTok | realTok | byteTok | boolenoTok

#### Regla 01

$\text{Prim}(\text{LlevaParamPrin1}) \cap \text{Prim}(\text{LlevaParamPrin2})$

$= \{\} \cap \text{Prim}(\text{ParamPrin})$

$= \{\} \cap \{\text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{boolenoTok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{LlevaParamPrin}) \cap \text{Sig}(\text{LlevaParamPrin})$

$= \{\text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{boolenoTok}\} \cap \{\text{corcheteF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “LlevaParamPrin”.

\*\*\*\*\*

**ParamPrin** → **TipoPrimIdent** **ParamPrinUno**

**ParamPrinUno** → ε | coma **TipoPrimIdent**

#### Regla 01

$\text{Prim}(\text{ParamPrinUno1}) \cap \text{Prim}(\text{ParamPrinUno2}) = \{\} \cap \{\text{coma}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{ParamPrinUno}) \cap \text{Sig}(\text{ParamPrinUno}) = \{\text{coma}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “ParamPrinUno”.

\*\*\*\*\*

**LlamadaMet** → **FuncionArreglo** | ident corcheteL **LlamadaMetUno** corcheteF

**LlamadaMetUno** → ε | **IdentExpres** **MasIdentExpres**

#### Regla 01

$\text{Prim}(\text{LlamadaMetUno1}) \cap \text{Prim}(\text{LlamadaMetUno2})$

$= \{\} \cap \text{Prim}(\text{IdentExpres})$

$= \{\} \cap \{\text{referencia}, \text{datoCaracter}, \text{ident}, \text{datoCadena}, \text{concatenarTok}, \text{verdaderoTok}, \text{falsoTok}, \text{negacion}\}$

$= \emptyset$

#### Regla 02

$\text{Prim}(\text{LlamadaMetUno}) \cap \text{Sig}(\text{LlamadaMetUno})$

$= \{\text{referencia}, \text{datoCaracter}, \text{ident}, \text{datoCadena}, \text{concatenarTok}, \text{verdaderoTok}, \text{falsoTok}, \text{negacion}\} \cap \{\text{corcheteF}\}$

$= \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**LlamadaMetUno**”.

\*\*\*\*\*

**LlamadaMetUno**  $\rightarrow \varepsilon \mid \text{IdentExpres MasIdentExpres}$

**MasIdentExpres**  $\rightarrow \varepsilon \mid \text{coma IdentExpres}$

#### Regla 01

$\text{Prim}(\text{MasIdentExpres1}) \cap \text{Prim}(\text{MasIdentExpres2}) = \{\} \cap \{\text{coma}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasIdentExpres}) \cap \text{Sig}(\text{MasIdentExpres}) = \{\text{coma}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasIdentExpres**”.

\*\*\*\*\*

**ExpresNum**  $\rightarrow \text{ExpresCon MasExpresNum}$

**MasExpresNum**  $\rightarrow \varepsilon \mid \text{otok ExpresCon}$

#### Regla 01

$\text{Prim}(\text{MasExpresNum1}) \cap \text{Prim}(\text{MasExpresNum2}) = \{\} \cap \{\text{otok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasExpresNum}) \cap \text{Sig}(\text{MasExpresNum}) = \{\text{otok}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasExpresNum**”.

\*\*\*\*\*

**ExpresCon**  $\rightarrow \text{ExpresRel MasExpresCon}$

**MasExpresCon**  $\rightarrow \varepsilon \mid \text{ytok ExpresRel}$

#### Regla 01

$\text{Prim}(\text{MasExpresCon1}) \cap \text{Prim}(\text{MasExpresCon2}) = \{\} \cap \{\text{ytok}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{MasExpresCon}) \cap \text{Sig}(\text{MasExpresCon}) = \{\text{ytok}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasExpresCon**”.

\*\*\*\*\*

**ExpresRel**  $\rightarrow \text{Negacion ExpresAri XexpresRel}$

**Negacion**  $\rightarrow \varepsilon \mid \text{negacion}$

#### Regla 01

$\text{Prim}(\text{Negacion1}) \cap \text{Prim}(\text{Negacion2}) = \{\} \cap \{\text{negacion}\} = \emptyset$

#### Regla 02

$\text{Prim}(\text{Negacion}) \cap \text{Sig}(\text{Negacion})$   
 $= \{\text{negacion}\} \cap \text{Prim}(\text{ExpresAri})$   
 $= \{\text{negacion}\} \cap \{\text{mas}, \text{menos}\} \cup \{\}$   
 $= \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “Negacion”.

\*\*\*\*\*

**ExpresRel**  $\rightarrow$  **Negacion** **ExpresAri** **XexpresRel**

**XexpresRel**  $\rightarrow \epsilon$  | **Simbolo** **ExpresAri**

**Simbolo**  $\rightarrow$  mnr | myr | mei | mai | igl | nig

#### Regla 01

$\text{Prim}(\text{XexpresRel1}) \cap \text{Prim}(\text{XexpresRel2})$   
 $= \{\} \cap \text{Prim}(\text{Simbolo})$   
 $= \{\} \cap \{\text{mnr}, \text{myr}, \text{mei}, \text{mai}, \text{igl}, \text{nig}\}$   
 $= \emptyset$

#### Regla 02

$\text{Prim}(\text{XexpresRel}) \cap \text{Sig}(\text{XexpresRel})$   
 $= \{\text{mnr}, \text{myr}, \text{mei}, \text{mai}, \text{igl}, \text{nig}\} \cap \{\}$   
 $= \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “XexpresRel”.

\*\*\*\*\*

**ExpresAri**  $\rightarrow$  **Termino** **MasExpresAri**

**MasExpresAri**  $\rightarrow \epsilon$  | **Signo** **Termino**

**Signo**  $\rightarrow$  mas | menos

#### Regla 01

$\text{Prim}(\text{MasExpresAri1}) \cap \text{Prim}(\text{MasExpresAri1})$   
 $= \{\} \cap \text{Prim}(\text{Signo})$   
 $= \{\} \cap \{\text{mas}, \text{menos}\}$   
 $= \emptyset$

#### Regla 02

$\text{Prim}(\text{MasExpresAri}) \cap \text{Sig}(\text{MasExpresAri}) = \{\text{mas}, \text{menos}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “MasExpresAri”.

\*\*\*\*\*

**Termino**  $\rightarrow$  **Factor** **MasTermino**

**MasTermino**  $\rightarrow \epsilon$  | **Oper** **Factor**



**Oper** → por | barra

Regla 01

$\text{Prim}(\text{MasTermino1}) \cap \text{Prim}(\text{MasTermino2})$   
 $= \{\} \cap \text{Prim}(\text{Oper})$   
 $= \{\} \cap \{\text{por}, \text{barra}\}$   
 $= \emptyset$

Regla 02

$\text{Prim}(\text{MasTermino}) \cap \text{Sig}(\text{MasTermino}) = \{\text{por}, \text{barra}\} \cap \{\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasTermino**”.

\*\*\*\*\*

**Factor** → **Sing Factores**

**Sing** → ε | **Signo**

**Signo** → mas | menos

Regla 01

$\text{Prim}(\text{Sing1}) \cap \text{Prim}(\text{Sing2})$   
 $= \{\} \cap \text{Prim}(\text{Signo})$   
 $= \{\} \cap \{\text{mas}, \text{menos}\}$   
 $= \emptyset$

Regla 02

$\text{Prim}(\text{Sing}) \cap \text{Sig}(\text{Sing})$   
 $= \{\text{mas}, \text{menos}\} \cap \text{Prim}(\text{Factores})$   
 $= \{\text{mas}, \text{menos}\} \cap \{\text{numeroEntero}, \text{numeroReal}, \text{parentI}, \text{ident}, \text{sistemaTok}\}$   
 $= \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**Sing**”.

\*\*\*\*\*

**FuncNum** → sistemaTok punto **Fnum** parentI **ExpresNum MasExpressNum** parentF

**MasExpressNum** → ε | coma **ExpresNum**

Regla 01

$\text{Prim}(\text{MasExpressNum1}) \cap \text{Prim}(\text{MasExpressNum2}) = \{\} \cap \{\text{coma}\} = \emptyset$

Regla 02

$\text{Prim}(\text{MasExpressNum}) \cap \text{Sig}(\text{MasExpressNum}) = \{\text{coma}\} \cap \{\text{parentF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasExpressNum**”.

\*\*\*\*\*

**FuncNumCad** → sistemaTok punto **Conlog** corcheteI **ValorCadena MasFuncNumCad** corcheteF

**MasFuncNumCad** → ε | coma **ValorCadena**

### Regla 01

$\text{Prim}(\text{MasFuncNumCad1}) \cap \text{Prim}(\text{MasFuncNumCad2}) = \{\} \cap \{\text{coma}\} = \emptyset$

### Regla 02

$\text{Prim}(\text{MasFuncNumCad}) \cap \text{Sig}(\text{MasFuncNumCad}) = \{\text{coma}\} \cap \{\text{corcheteF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasFuncNumCad**”.

\*\*\*\*\*

**Bloque**  $\rightarrow$  llaveI **MasBloque Instruccion** llaveF

**MasBloque**  $\rightarrow \epsilon$  | **Dedvar** puntoycoma **MasBloque**

### Regla 01

$\text{Prim}(\text{MasBloque1}) \cap \text{Prim}(\text{MasBloque2})$

$= \{\} \cap \text{Prim}(\text{Dedvar})$

$= \{\} \cap \{\text{ident}, \text{objetoTok}, \text{archivoTok}, \text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{booleanoTok}, \text{arregloTok}\}$

$= \emptyset$

### Regla 02

$\text{Prim}(\text{MasBloque}) \cap \text{Sig}(\text{MasBloque})$

$= \{\text{ident}, \text{objetoTok}, \text{archivoTok}, \text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{booleanoTok}, \text{arregloTok}\} \cap \text{Prim}(\text{Instruccion})$

$= \{\text{ident}, \text{objetoTok}, \text{archivoTok}, \text{caracterTok}, \text{cadenaTok}, \text{enteroTok}, \text{realTok}, \text{byteTok}, \text{booleanoTok}, \text{arregloTok}\} \cap \{\text{romperTok}, \text{retornarTok}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{siTok}, \text{probarTok}, \text{mientrasTok}, \text{paraTok}, \text{hacerTok}, \text{paracadaTok}, \text{sistemaTok}\} \cup \{\}$

$= \{\text{ident}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**MasBloque**”.

\*\*\*\*\*

**Instruccion**  $\rightarrow \epsilon$  | **MasInstruccion Instruccion**

**MasInstruccion**  $\rightarrow$  **IdentLlamada** puntoycoma | **LlamadaMet** puntoycoma | **Asignacion** puntoycoma | romperTok puntoycoma | **InstSi** | **InstProbar** | **InstMientras** | **InstPara** | **InstHacer** | **InstParaCad** | **InstES** | retornarTok **Expresion** puntoycoma

### Regla 01

$\text{Prim}(\text{Instruccion1}) \cap \text{Prim}(\text{Instruccion2})$

$= \{\} \cap \text{Prim}(\text{MasInstruccion})$

$= \{\} \cap \{\text{romperTok}, \text{retornarTok}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{siTok}, \text{probarTok}, \text{mientrasTok}, \text{paraTok}, \text{hacerTok}, \text{paracadaTok}, \text{sistemaTok}\}$

$= \emptyset$

PERO:

$\text{Prim}(\text{IdentLlamada}) = \{\text{ident}\}$

$\text{Prim}(\text{LlamadaMet}) = \{\text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}\}$

$\text{Prim}(\text{Asignacion}) = \{\text{ident}\}$

$\text{Prim}(\text{InstES}) = \{\text{sistemaTok}, \text{ident}\}$

### Regla 01

$\text{Prim}(\text{IdentLlamada}) \cap \text{Prim}(\text{LlamadaMet}) \cap \text{Prim}(\text{Asignacion}) \cap \text{Prim}(\text{InstES}) = \{\text{ident}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**Instruccion**”.

Semántica: Si se encuentra un identificador, se verifica si es un método, un objeto, un objeto de tipo archivo o una variable, para decidir si se trata de una llamada a un método, un objeto a utilizar, un objeto archivo para llamada de método o una asignación a una variable.

\*\*\*\*\*

**Expresion** → **ValorCaracter** | **ValorCadena** | **DatoBool** | **ExpresNum**

$\text{Prim}(\text{ValorCaracter}) = \{\text{datoCaracter}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}\}$

$\text{Prim}(\text{ValorCadena}) = \{\text{datoCadena}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{concatenarTok}\}$

$\text{Prim}(\text{DatoBool}) = \{\text{verdaderoTok}, \text{falsoTok}\}$

$\text{Prim}(\text{ExpresNum}) = \{\text{negacion}, \text{mas}, \text{menos}, \text{numeroEntero}, \text{numeroReal}, \text{parentl}, \text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}, \text{sistemaTok}\}$

### Regla 01

$\text{Prim}(\text{ValorCaracter}) \cap \text{Prim}(\text{ValorCadena}) \cap \text{Prim}(\text{DatoBool}) \cap \text{Prim}(\text{ExpresNum})$

$= \{\text{ident}, \text{agregarTok}, \text{obtenerTok}, \text{quitarTok}, \text{cuantosTok}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**Expresion**”.

Semántica: Si se encuentra un identificador la compilación seguirá si se cumple lo siguiente:

- Si el identificador es una variable entera o real entonces seguirá por el camino de Expresión\_Numerica.
- Si el identificador es una función que retorna un valor entero o real entonces seguirá por el camino de Expresión\_Numerica.
- Si el identificador es una variable de tipo carácter o una función que retorna un valor de tipo carácter entonces seguirá por el camino de ValorCaracter.
- Si el identificador es una variable de tipo cadena o una función que retorna un valor de tipo cadena entonces seguirá por el camino de ValorCadena.

Si se encuentra una palabra reservada agregar, obtener, quitar, cuantos, entonces se seguirá por el camino de Función\_Arreglo.

\*\*\*\*\*

**InstSiUno** → coma **Asignacion** coma **Asignacion** corcheteF puntoycoma | corcheteF llaveF **Instruccion**

**InstSiDos** llaveF

**InstSiDos** → ε | llaveF sinoTok llaveF **Instruccion**

### Regla 01

$\text{Prim}(\text{InstSiDos1}) \cap \text{Prim}(\text{InstSiDos2}) = \{\} \cap \{\text{llaveF}\} = \emptyset$

### Regla 02

$\text{Prim}(\text{InstSiDos}) \cap \text{Sig}(\text{InstSiDos}) = \{\text{llaveF}\} \cap \{\text{llaveF}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**InstSiDos**”.

Semántica: se deberá leer el siguiente token, si es un “sino” se debe realizar la producción InstSiDos sino no y continua.

\*\*\*\*\*

**InstProbar** → probarTok corcheteI **Expresion** corcheteF llaveI casoTok **Expresion** dospuntos **Instruccion**  
**MasInstProbar InstProbarUno** llaveF

**InstProbarUno** → ε | defectoTok dospuntos **Instruccion**

Regla 01

$\text{Prim}(\text{InstProbarUno1}) \cap \text{Prim}(\text{InstProbarUno2}) = \{\} \cap \{\text{defectoTok}\} = \emptyset$

Regla 02

$\text{Prim}(\text{InstProbarUno}) \cap \text{Sig}(\text{InstProbarUno}) = \{\text{defectoTok}\} \cap \{\text{llaveF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**InstProbarUno**”.

\*\*\*\*\*

**InstProbar** → probarTok corcheteI **Expresion** corcheteF llaveI casoTok **Expresion** dospuntos **Instruccion**  
**MasInstProbar InstProbarUno** llaveF

**InstProbarUno** → ε | defectoTok dospuntos **Instruccion**

**MasInstProbar** → ε | casoTok **Expresion** dospuntos **Instruccion**

Regla 01

$\text{Prim}(\text{MasInstProbar1}) \cap \text{Prim}(\text{MasInstProbar1}) = \{\} \cap \{\text{casoTok}\} = \emptyset$

Regla 02

$\text{Prim}(\text{MasInstProbar}) \cap \text{Sig}(\text{MasInstProbar})$

$= \{\text{casoTok}\} \cap \text{Prim}(\text{InstProbarUno})$

$= \{\text{casoTok}\} \cap \{\text{defectoTok}\} \cup \{\}$

$= \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**MasInstProbar**”.

\*\*\*\*\*

**InstMostrar** → sistemaTok punto mostrarTok corcheteI **Expresion InstMostrarUno** corcheteF  
puntoycoma

**InstMostrarUno** → ε | coma **Expresion**

Regla 01

$\text{Prim}(\text{InstMostrarUno1}) \cap \text{Prim}(\text{InstMostrarUno2}) = \{\} \cap \{\text{coma}\} = \emptyset$

Regla 02

$\text{Prim}(\text{InstMostrarUno}) \cap \text{Sig}(\text{InstMostrarUno}) = \{\text{coma}\} \cap \{\text{corcheteF}\} = \emptyset$

Por tanto, la gramática CUMPLE con ser LL(1) en la producción “**InstMostrarUno**”.

\*\*\*\*\*

**Factor** → **Sing Factores**

**Sing**  $\rightarrow \varepsilon$  | **Signo**

**Factores**  $\rightarrow$  numeroEntero | numeroReal | **IdentLlamada** | **LlamadaMet** | **FuncNum** | **FuncNumCad** | parentI **ExpresNum** parentF

Prim(IdentLlamada) = {ident}

Prim(LlamadaMet) = {ident, agregarTok, obtenerTok, quitarTok, cuantosTok}

Prim(FuncNum) = {sistemaTok}

Prim(FuncNumCad) = {sistemaTok}

#### Regla 01

$\text{Prim(IdentLlamada)} \cap \text{Prim(LlamadaMet)} = \{\text{ident}\} \neq \emptyset$

$\text{Prim(FuncNum)} \cap \text{Prim(FuncNumCad)} = \{\text{sistemaTok}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**Factores**”.

Semántica: Si se encuentra un identificador la compilación seguirá si se cumple lo siguiente:

- Si el identificador es una variable entera o real entonces seguirá por el camino de Declaración\_Variable (atributo).
- Si el identificador es un método entonces seguirá por el camino de Llamada\_Metodo.

Si es la palabra reservada “Sistema” se adelantará dos tokens hasta determinar a cuál función hace referencia, si es del camino Funcion\_Numerica o Funcion\_Numerica\_Cadena.

\*\*\*\*\*

**ValorCaracter**  $\rightarrow$  datoCaracter | **IdentLlamada** | **LlamadaMet**

#### Regla 01

$\text{Prim(IdentLlamada)} \cap \text{Prim(LlamadaMet)} = \{\text{ident}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**ValorCaracter**”.

Semántica: Si se encuentra un identificador la compilación seguirá si se cumple lo siguiente:

- Si el identificador es una variable de tipo carácter entonces seguirá por el camino de Declaración\_Variable (atributo).
- Si el identificador es una función que retorna un carácter entonces seguirá por el camino de Llamada\_Metodo.
- Si el identificador es cualquier otra cosa entonces se emitirá un error.

\*\*\*\*\*

**ValorCadena**  $\rightarrow$  datoCadena | **IdentLlamada** | **LlamadaMet** | **FuncCad**

#### Regla 01

$\text{Prim(IdentLlamada)} \cap \text{Prim(LlamadaMet)} = \{\text{ident}\} \neq \emptyset$

Por tanto, la gramática NO CUMPLE con ser LL(1) en la producción “**ValorCaracter**”.

Semántica: Si se encuentra un identificador la compilación seguirá si se cumple lo siguiente:

- Si el identificador es una variable de tipo cadena entonces seguirá por el camino de Declaración\_Variable (atributo).
- Si el identificador es una función que retorna una cadena entonces seguirá por el camino de Llamada\_Metodo.

Si es la palabra reservada “concatenar” entonces seguirá por el camino de Función\_Cadena.

\*\*\*\*\*

## O. Reglas Semánticas Extras Aplicadas

En la producción **Bloque**.

- ❖ Si el bloque pertenece a un método de tipo función debe existir al menos un “retornar” con su respectiva expresión.
- ❖ Si el bloque pertenece a un método de tipo procedimiento no puede existir un “retornar”.

En la producción **Llamada\_Metodo**.

- ❖ Verificar si el identificador se encuentra declarado como un método.
- ❖ Verificar que la cantidad de parámetros del método sea igual a la definida en su declaración
- ❖ Verificar que el valor de los parámetros corresponda al tipo con el que fueron declarados.  
(Validación realizada en tiempo de ejecución).

En la producción **Asignacion**.

- ❖ Luego de evaluar Expresión se debe comprobar que exista una compatibilidad de tipos  
(Validación realizada en tiempo de ejecución).

En la producción **Instruccion\_Para**.

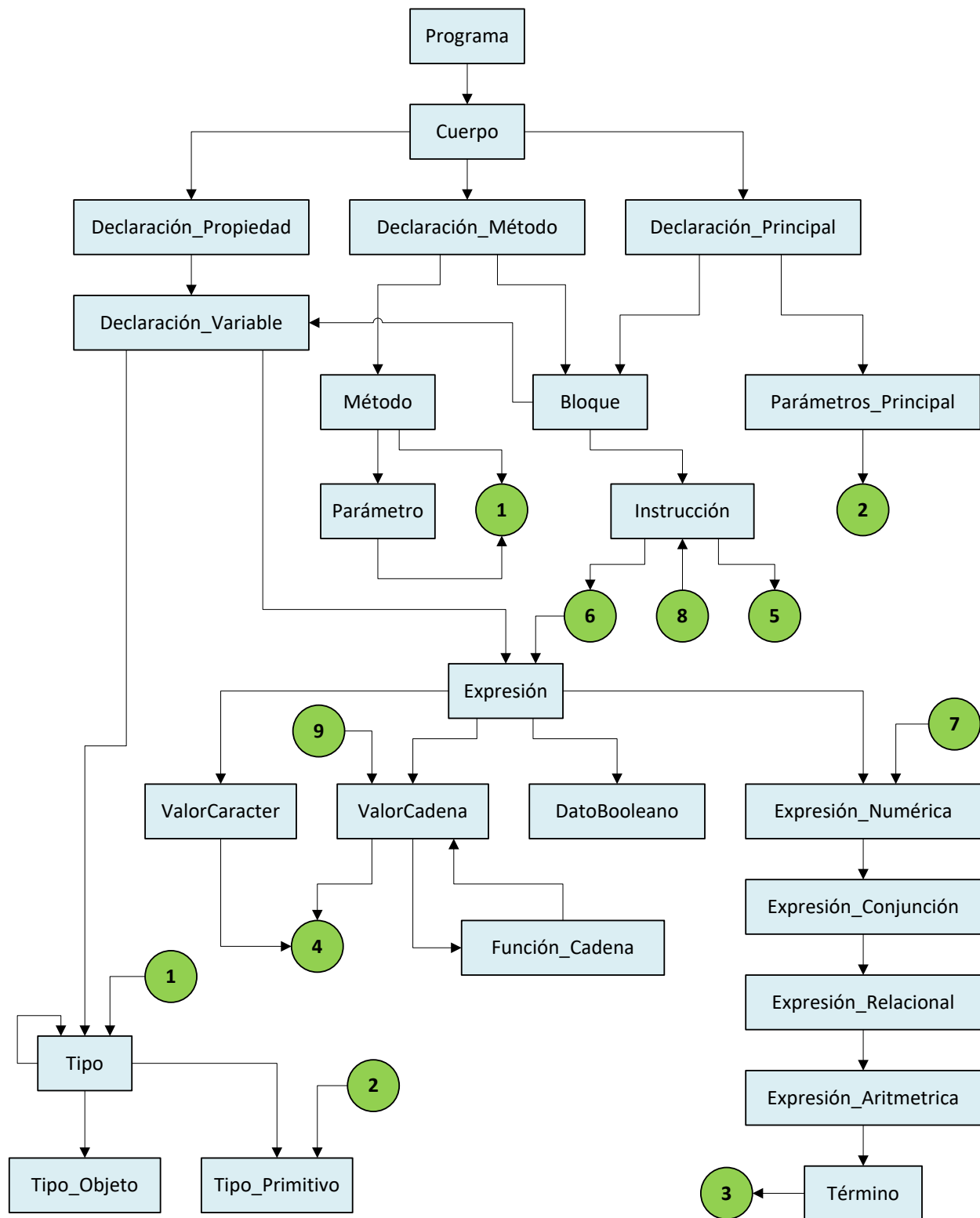
- ❖ Variable debe ser un variable entera.
- ❖ El valor de Expresion\_Numerica debe ser un entero o al menos que exista compatibilidad de tipos.
- ❖ (Validación realizada en tiempo de ejecución).
- ❖ Para
  - » Limite1: Valor de la primera aparición de Expresion\_Numerica.
  - » Limite2: Valor de la segunda aparición de Expresion\_Numerica.
  - » Paso: Valor de la tercera aparición de Expresion\_Numerica.

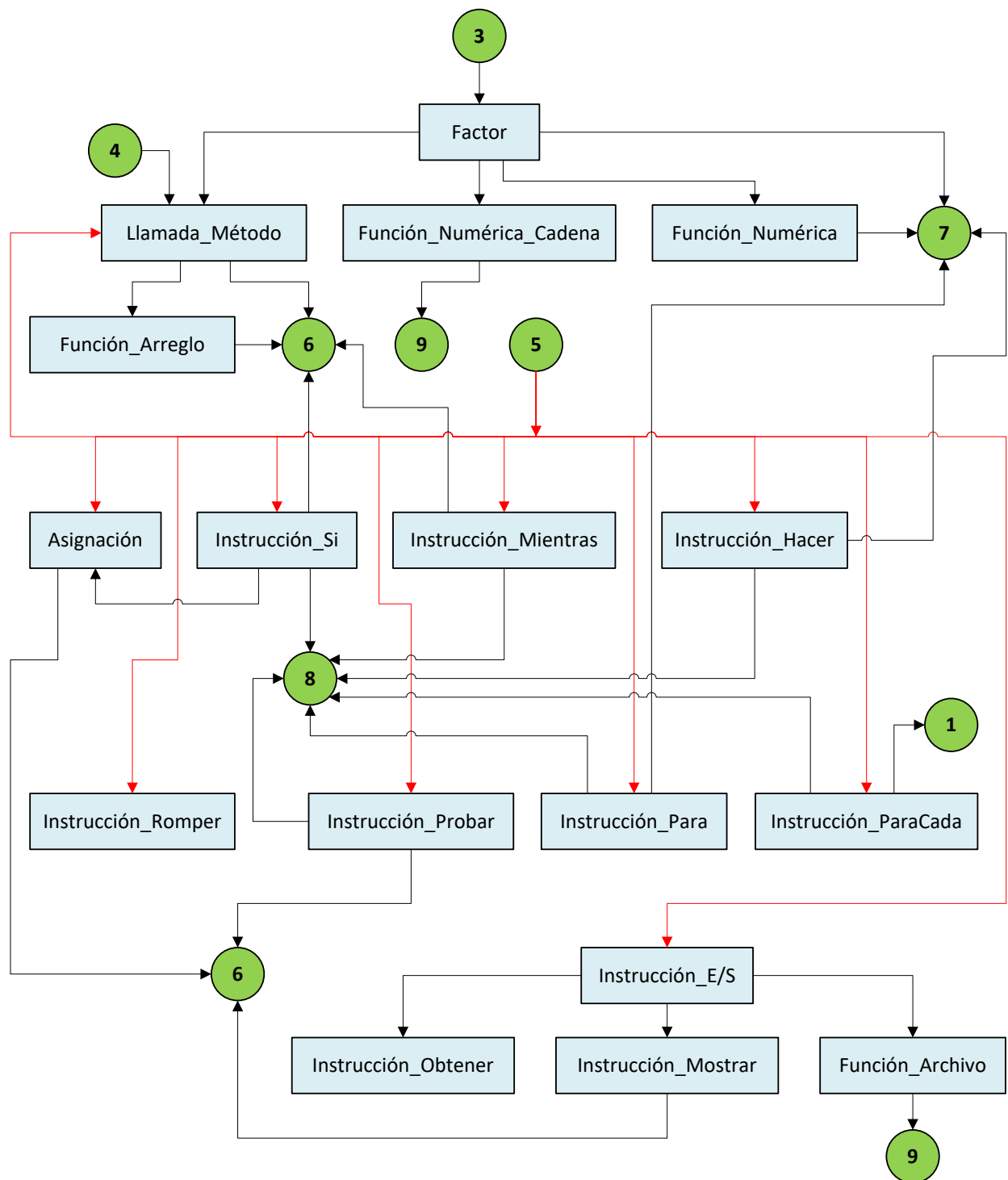
Se debe cumplir una de las siguientes dos reglas:

- a) Si  $\text{Limite1} < \text{Limite2}$  entonces  $\text{Paso} > 0$ .
- b) Si  $\text{Limite1} > \text{Limite2}$  entonces  $\text{Paso} < 0$ .

(Validación realizada en tiempo de ejecución)

## P. Diagrama de Dependencias







## Q. Estrategias para la estabilización de errores sintácticos y semánticos

- ❖ En la función del analizador sintáctico del compilador cada llamada a alguna otra función, se añadirá un parámetro extra que contendrá un conjunto de tokens seguidores denominado toksig, el cual contendrá todas las palabras reservadas o caracteres especiales del lenguaje que se esperan ver después de que la función invocada finalice el análisis sintáctico correspondiente.  
Esto con el fin de que dicho conjunto toksig ante la presencia de un error en la función llamada, se logre realizar un salto de texto (o código) hasta encontrar un token con el que la función (la que realizó inicialmente la llamada de la secuencia) pueda continuar analizando el código fuente.
- ❖ En la función del analizador sintáctico del compilador cada llamada a alguna otra función, agregará el parámetro toksig de la función invocada su propio conjunto toksig. Esto para que, al encontrar un error en la función llamada, se podrá también realizar un salto de código (texto) hasta un token con el que la función precedente a la función que realizó la invocación pueda usar.
- ❖ Antes de iniciar el código del analizador sintáctico se crea un conjunto toksig de arranque (conjunto inicial), que tenga los elementos que siguen de la primera función del análisis sintáctico, de manera que pueda iniciarse la sincronización encargada de estabilizar la detección de errores dentro del compilador.
- ❖ Cuando se encuentra un error se realiza un salto de texto inteligente (buscando el token del conjunto toksig al cual pueda saltar), aprobando que el proceso de compilación pueda continuar y evitando la generación de errores en cascada.
- ❖ Como cada archivo \*.acl puede contener más de una clase, el conjunto de arranque (conjunto inicial) está compuesto de tres tokens: publicaTok, localTok, claseTok, dichos tokens corresponden a las primeras palabras reservadas que busca en el archivo y que pueden generar errores dentro de su sintaxis; el conjunto toksig para este caso es únicamente el token llaveF.

## R. Estrategia para el Manejo de Código Intermedio (Múltiples Archivos)

Cada archivo .acl se crea un archivo de código intermedio con la extensión .ob y un archivo extra para guardar una estructura donde estén los valores de cada atributo de la clase .strc así:

Clase1.acl	Clase1.ob	Clase1.strc
Contiene el código fuente del programa	Contiene el código intermedio del programa (código-p)	Estructura con los valores de cada atributo de la clase.

Cada **clase** utiliza una plantilla y cada instancia tiene que tener la misma forma. Así por cada clase se genera un código intermedio completo.

Persona.acl	Persona.ob	Persona.strc	Main.acl	Main.ob	Main.strc

## S. Diseño del Código Intermedio

El código intermedio generado por el compilador del lenguaje Armus cuenta con los siguientes mnemónicos para la construcción de sus instrucciones:

- LIT** Cargar un valor literal al tope de la pila. Este valor literal puede ser un número entero, un número real, un carácter o una cadena.
- SAL** Salto incondicional