Compiladores 2021-1

Prof. Adrián Ulises Mercado Martínez

Proyecto Final

Coronel Ruiz Aldair	García Pérez Adrián	Toporek Coca Eric
31512683-1	31513122-4	31428498-7

Introducción.

El diseño del front-end para un compilador cubre las siguientes fases:

- Diseño de Expresiones regulares que definen la gramática del lenguaje.
- Análisis Léxico y generación de tokens del lenguaje.
- Análisis Sintáctico.
- Análisis Semántico.
- Generación de código intermedio.

A partir del material proporcionado por el Profesor (la gramática proporcionada a continuación) se pudo ir construyendo paso a paso lo que consta el mismo front-end a partir de los contenidos vistos en el curso de Compiladores.

PRODUCCIÓN
programa → declaraciones funciones
declaraciones → tipo lista_var ; declaraciones | ε
tipo → basico compuesto
basico → **int** | **float** | **char** | **double**| **void**
compuesto → **(numero)** compuesto | ε
lista_var → lista_var , **id** | **id**
funciones → **func** tipo **id (** argumentos **)** bloque funciones | ε
argumentos → lista_args | ε
lista_args → lista_args, tipo **id** | tipo **id**
bloque → { declaraciones instrucciones }
instrucciones → instrucciones sentencia | sentencia
sentencia → parte_Izquierda **=** bool **;**| **if(** bool **)** sentencia
| **if(** bool **)** sentencia **else** sentencia | **while(** bool **)** sentencia
| **do** sentencia **while(** bool **)** | **break ;** | bloque | **return** exp **;**| **return;**
| **switch(** bool **)** { casos }| **print** exp **;** | **scan** parte_Izquierda
casos → caso casos| ε | predeterminado
caso → **case numero:** instrucciones
predeterminado→ **default:** instrucciones
parte_Izquierda → **id** localizacion |**id**
bool → bool || comb | comb
comb → comb **&&** igualdad | igualdad
igualdad → igualdad **==** rel | igualdad **!=** rel | rel
rel → exp < exp | exp <= exp | exp >= exp |
| exp > exp | exp
exp → exp + term | exp – term | term
term → term * unario | term / unario | term **%** unario | unario
unario → !unario | – unario | factor
factor → **(**bool**)** | **id** localizacion | **numero** | **cadena** | **true**
| **false** | **id(**parametros**)** | **id**
parametros → lista_param | ε
lista_param → lista_param , bool | bool
localizacion→ localizacion **[** bool **]** | **[** bool **]**

Análisis Léxico

Primero, se tienen que identificar todos los tokens de programa, es decir, los símbolos terminales que fueron resaltados en la gramática proporcionada y darles algún valor numérico que lo identifique como token, siendo los tokens definidos como sigue:

```
COMA 1001 // ,
PCOMA 1002 // ;
CIZQ 1003 // [
CDER 1004 // ]
INT 1005  // int
FLOAT 1006 // float
NUM 1007 // double
ID 1008 // id

PIZQ 1009 // (
PDER 1010 // )
CHAR 1011 // char
DOUBLE 1012  // double
VOID 1013 // void
IF 1014 // if
ELSE 1015 // else
DO 1016 // do
WHILE 1017 // while
BREAK 1018 // break
SWITCH 1019 // switch
DEFAULT 1020 // default
CASE 1021 // case
```

```
AND 1024 // &&
EQ 1026 // ==
NEQ 1027 // !=
LESS 1028 // <
LEQ 1029 // <=
GEQ 1030 // >=
GREATER 1031 // >
PLUS 1032 // +
MINUS 1033 // -
MULT 1034 // *
DIV 1035 // /
MOD 1036 // %
NOT 1037 // !


STR 1035 // cadena
DDOT 1036 // :
ASIG 1037 // =
OR 1039 // ||
LKEY 1040 // {
RKEY 1041 // }

TRUE 1042
FALSE 1043

FUNC 1044
RETURN 1045
PRINT 1046
SCAN 1047

FIN 9001 // $
```

Para la implementación del análisis léxico, primero fue definida una clase token, que guarda la clase y el valor de la expresión según sea leída, y nos respaldamos en **lexer** que según sea lo que se lea puede generar un token, ignorarlo (whitespcs y comentarios) o mandar un error en caso de que no se proporcione algo definido por la gramática. Los detalles de implementación sobre el lexer fueron cubiertos en la práctica 5.

Análisis Sintáctico

Diseño de expresiones regulares.

A partir de la gramática dada, para simplificar las expresiones regulares por en typos más sencillos, se escogió en orden una letra del alfabeto para la asignación. Quedando de la siguiente forma:

programa	=	A
declaraciones	=	B
tipo	=	C

básico	=	D
compuesto	=	E
lista_var	=	F
funciones	=	G
argumentos	=	H
lista_args	=	I
bloque	=	J
instrucciones	=	K
sentencia	=	L
casos	=	M
caso	=	N
predeterminado	=	O
parte_izq	=	P
bool	=	Q
comb	=	R
igualdad	=	S
rel	=	T
exp	=	U
term	=	V
unario	=	W
factor	=	X
parámetros	=	Y
lista_param	=	Z
localización	=	AA

Por lo tanto, la gramática quedaría redefinida cómo sigue:

Gramática:

A ->	B G						
B ->	C F;B	ε					
C ->	D E						
D ->	**int**	**float**	**char**	**double**	**void**		
E ->	**[numero]** E	ε					
F ->	**F, id**	**id**					
G ->	**func** C **id(** H **)** J G		ε				
H ->	I	ε					
I ->	**id**	**I , C id**	ε				
J ->	**{** B K **}**						
K ->	K L	L					
L ->	P = Q;	**if(**Q**)** L	**if(**Q**)** L **else** L	**while(**Q**)** L	**do** L **while(**Q**)**	**break;**	J
	return U;	**switch(**Q**) {**M**}**	**print** U;	**scan** P	**return;**		
M ->	N M	O	ε				
N ->	**case numero:** K						
O ->	**default:** K						
P ->	**id** AA	**id**					
Q ->	Q **||** R	R					
R ->	R **&&** S	S					
S ->	S' **==** T	S **!=** T	T				
T ->	U **<** U	U **<=** U	U **>=** U	U **>** U	U		
U ->	U **+** V	U **-** V	V				
V ->	V ***** W	V **/** W	V **%** W	W			
W ->	**!**W	**-**W	X				
X ->	**(**Q**)**	**id** X'	**numero**	**cadena**	**true**	**false**	
Y ->	Z	ε					
Z ->	**Z,**Q	Q					
AA ->	**AA [**Q**]**	**[**Q**]**					

Notemos que las producciones **F, K, Q, R, S, U, V, Z** y **AA** son **recursivas por la izquierda**, por lo que al quitar la misma sobre las producciones mencionadas anteriormente se tiene lo siguiente:

Gramática:

A ->	B G						
B ->	C F;B	ε					
C ->	D E						
D ->	**int**	**float**	**char**	**double**	**void**		
E ->	**[numero]** E	ε					
F ->	**id** F'						
F' ->	**, id** F'	ε					
G ->	**func** C **id(** H **)** J G		ε				
H ->	I	ε					
I ->	C **id** I'	ε					
I' ->	**, C id** I'	ε					
J ->	**{** B K **}**						
K ->	L K'						
K' ->	L K'	ε					
L ->	P = Q;	**if(**Q**)** L	**if(**Q**)** L **else** L	**while(**Q**)** L	**do** L **while(**Q**)**	**break;**	J
	return U;	**switch(**Q**) {**M**}**	**print** U;	**scan** P	**return;**		
M ->	N M	O	ε				
N ->	**case numero:** K						
O ->	**default:** K						
P ->	**id** AA	**id**					
Q ->	R Q'						
Q' ->	**||** R Q'	ε					
R ->	S R'						
R' ->	**&&** S R'	ε					
S ->	T S'						

S' ->	== T S'	!= T S'	ε					
T ->	U < U	U <= U	U >= U	U > U	U			
U ->	V U'							
U' ->	+ V U'	- V U'	ε					
V ->	W V'							
V' ->	* W V'	/ W V'	% W V'	ε				
W ->	!W	-W	X					
X ->	(Q)	**id** AA	**numero**	**cadena**	**true**	**false**	**id**(Y)	**id**
Y ->	Z	ε						
Z ->	Q Z'							
Z' ->	,Q Z'	ε						
AA ->	**[**Q] AA'							
AA' ->	**[**Q] AA'	ε						

Finalmente, notemos que en la producción **L** se encuentran dos factores comunes por la izquierda, además en la producción **T**,**P** y **X**, por lo que al factorizar, la gramática definitiva está definida cómo sigue:

Gramática:								
A ->	B G							
B ->	C F;B	ε						
C ->	D E							
D ->	**int**	**float**	**char**	**double**	**void**			
E ->	**[numero]** E	ε						
F ->	**id** F'							
F' ->	**, id** F'	ε						
G ->	**func** C **id(** H **)** J G		ε					
H ->	I	ε						
I ->	C **id** I'	ε						
I' ->	**,** C **id** I'	ε						
J ->	**{** B K **}**							
K ->	L K'							
K' ->	L K'	ε						
L ->	P = Q;	**if(**Q**)** L L'	**while(**Q**)** L	**do** L **while(**Q**)**	**break;**	J		
	return L''	**switch(**Q**) {**M**}**	**print** U;	**scan** P				
L' ->	**else** L	ε						
L'' ->	U;	;						
M ->	N M	O	ε					
N ->	**case numero:** K							
O ->	**default:** K							
P ->	**id** P'							
P' ->	AA	ε						
Q ->	R Q'							
Q' ->	|| R Q'	ε						
R ->	S R'							
R' ->	**&&** S R'	ε						
S ->	T S'							
S' ->	== T S'	!= T S'	ε					
T ->	U T'							
T' ->	< U	<= U	>= U	> U	ε			
U ->	V U'							
U' ->	+ V U'	- V U'	ε					
V ->	W V'							
V' ->	* W V'	/ W V'	% W V'	ε				
W ->	!W	-W	X					
X ->	(Q)	**id** X'	**numero**	**cadena**	**true**	**false**		
X' ->	AA	(Y)	ε					
Y ->	Z	ε						
Z ->	Q Z'							
Z' ->	,Q Z'	ε						
AA ->	**[**Q] AA'							
AA' ->	**[**Q] AA'	ε						

Definición Dirigida Por Sintaxis

Basándonos en la gramática dada anteriormente, solo resta adecuar las reglas semánticas a la primera gramática generada (es decir, la que traduce a letras los símbolos no terminales). Por lo tanto, tenemos:

Producción	Regla Semántica
A -> B G	PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0
B -> C F; B	F.tipo = C.tipo
B -> epsilon	
C -> D E	E.base = D.tipo C.tipo = E.tipo
D -> **int**	D.tipo = **int**
D -> **float**	D.tipo = **float**
D -> **char**	D.tipo = **char**
D -> **double**	D.tipo = **double**
D -> **void**	D.tipo = **void**
E -> **[numero]** E1	E.tipo = PilaTT.top().insertar("array", num.val, E1.tipo) E1.base = E.base
E -> epsilon	E.tipo = E.base

F -> **id** F'	F'.tipo = F.tipo **Si !** PilaTS.top().buscar(id) **Entonces** PilaTS.top().insertar(id, F.tipo, dir, "var", NULO)) dir = dir + PilaTT.top().getTam(C.tipo) **SiNo** error ("El id no está declarado") **FinSi**
F' -> **,** **id** F'1	F'1.tipo = F'.tipo **Si !** PilaTS.top().buscar(id) **Entonces** PilaTS.top().insertar(id, F'.tipo, dir, "var", NULO)) dir = dir + PilaTT.top().getTam(C.tipo) **SiNo** error ("El id no está declarado") **FinSi**
F' -> epsilon	
G -> **func** C **id** **(** H **)** I G	ListaRetorno = NULL PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos) PilaDir.push(dir) dir = 0 **Si !** PilaTS.top().buscar(id) **Entonces** **Si** equivalentesLista(ListaRetorno, C.tipo) **Ent** PilaTS.top().insertar(id,C.tipo,-,`func` H.lista) genCod(label(id)) I.sig = nuevaEtq() genCod(label(I.sig)) **Sino** error("Retorno no coincide") **FinSi** **Sino** error("ID no declarado") **FinSi** PilaTS.pop() PilaTT.pop() dir = PilaDir.pop()
G -> epsilon	
H -> I	H.lista = I.lista
H -> epsilon	H.lista = NULL
I -> C **id** I'	**Si !** PilaTS.top().buscar(id) **Entonces** PilaTS.top().insertar(id, C.tipo, dir, "param",NULL) dir = dir + PilaTT.top().getTam(C.tipo) **Sino** error ("ID no está declarado") **FinSi** I.lista = I'.lista I.lista.agregar(C.tipo)
I' -> **,** C **id** I'1	**Si !** PilaTS.top().buscar(id) **Entonces** PilaTS.top().insertar(id, C.tipo, dir, "param",NULL) dir = dir + PilaTT.top().getTam(C.tipo) **Sino** error ("ID no está declarado") **FinSi** I'.lista = I'1.lista I'.lista.agregar(C.tipo)
I' -> epsilon	I'.lista = nuevaListaArgs() I'.lista.agregar(C.tipo)
J -> **{** B K **}**	K.sig = J.sig genCod(label(K.sig)
K -> L K'	K'.sigH = K.sig L.sig = nuevaEtq() genCode(label(L.sig)
K' -> L K'1	L.sig = nuevaEtq() genCod(label(L.sig)) K'1.sigH = K'.sigH K'.sig = K'1.sig
K' -> epsilon	K'.sig = K'.sigH
L -> P = Q **;**	**Si** equivalentes(P.tipo, Q.tipo) **Entonces** d1 = reducir(Q,dir, Q.tipo, P.tipo) genCod(P.dir `=` d1) **Sino** error("Tipos incompatibles") **FinSi**
L -> **if(** Q **)** L1 L'	Q.vddr = nuevaEtq() Q.fls = nuevoIndice() L1.sig = L.sig L'.sig = L.sig L'.lista_indices = nuevaListaIndices() L'.lista_indices.agregar(Q.fls) genCod(label(Q.vddr))
L' -> **else** L	L.sig = L'.sig genCod('goto' L.sig) genCod(label(L'.lista_indices[0])) remplazarIndices(L'.lista_indices,nuevaEtiqueta(),cuadruplas)
L' -> epsilon	reemplazarIndices(L'.lista_indices,L'.sig,cuadruplas)
L -> **while(** Q **)** L1	L1.sig = nuevaEtq() Q.vddr = nuevaEtq() Q.fls = L.sig genCod(label(L1.sig)) genCod(label(Q.vddr)) genCod(`goto` L1.sig)
L -> **do** L1 **while(** Q **)**	L1.sig = nuevaEtq() Q.vddr = nuevaEtq() Q.fls = L.sig genCod(label(L1.sig)) genCod(label(Q.vddr))
L -> **break;**	genCod(goto L.sig)
L -> J	J.sig = L.sig
L -> **return** L''	
L'' -> U**;**	ListaRetorno.agregar(**void**) genCod(`return`)
L'' -> **;**	ListaRetorno.agregar(**void**) genCod(`return`)

L -> **switch(** Q **) { M }**	M.etqPrueba = nuevaEtq() genCode(`goto` M.etqPrueba) M.sig = L.sig M.id = Q.dir genCode(label(M.etqPrueba)) genCode(M.prueba)
L -> **print** U;	genCod(`print` U.val)
L -> **scan** P	genCod(`scan` P.dir)
M -> N M1	M1.sig = M.sig N.sig = M.sig M.prueba = M.prueba || M1.prueba
M -> O	M.prueba = O.prueba O.sig = M.sig
M -> epsilon	M.prueba = ''
N -> **case num**: K	N.inicio = nuevaEtq() K.sig = M.sig N.prueba = genCod(**if** N.id `==` **num.lexval** `goto` M.inicio) genCode(label(N.inicio))
O -> **default:** K	O.inicio = nuevaEtq() K.sig = O.sig O.prueba = genCod(`goto` O.inicio) genCod(label(O.inicio))
P -> **id** P'	P'.base = **id**.lexval P.dir = P'.dir P.tipo = P'.tipo
P' -> AA	AA.base = P'.base P'.dir = AA.dir P'.tipo = AA.tipo
P' -> epsilon	**Si** PilaTS.top().buscar(P'.base) **Entonces** P'.dir = P'.base P'.tipo = PilaTS.top().getTipo(P'.dir) **Sino** error("El id no está declarado") **FinSi**
Q -> R Q'	R.vddr = Q.vddr R.fls = nuevoIndice() Q'.tipoH = R.tipo Q'.lista_indices = nuevaListaIndices() Q'.lista_indices.agregar(R.fls) Q.tipo = Q'.tipoS genCod(label(R.fls))
Q' -> || R Q'	**Si** equivalentes(Q'.tipoH, R.tipo) **Entonces** R.vddr = Q.vddr R.fls = nuevoIndice() Q'1.tipoH = R.tipo Q'1.vddr = Q.vddr Q'1.fls = Q.fls Q'1.lista_indices = Q'1.lista_indices Q'1.lista_indices.agregar(R.fls) Q'.tipoS = Q'1.tipoS genCod(label(Q1.fls)) **Sino** error("Tipos incompatibles") **FinSi**
Q' -> epsilon	reemplazarIndices(Q'.lista_indices, Q'.fls, cuadruplas) Q'.tipoS = int
R -> S R'	S.vddr = nuevoIndice() S.fls = R.fls R'.tipoH = S.tipo R'.lista_indices = nuevaListaIndices() R'.lista_indices.agregar(S.vddr) R.tipo = R'.tipoS genCod(label(S.vddr))
R' -> **&&** S R'	**Si** equivalentes(R'.tipoH, S.tipo) **Entonces** S.vddr = nuevoIndice() S.fls = R'.fls (aunque dice R, no R') R'1.tipoH = R'.tipo (lo mismo) R'1.vddr = bool.vddr R'1.fls = bool.fls R'1.lista_indices = bool'1.lista_indices R'1.lista_indices.agregar(S.vddr) R'.tipoS = R'1.tipoS genCod(label(S.vddr)) **Sino** error("Tipos incompatibles") **FinSi**
R' -> epsilon	reemplazarIndices(R'.lista_indices, R'.vddr, cuadruplas) R'.tipoS = **INT**
S -> T S'	T.vddr = S.vddr T.fls = S.fls S'.vddr = S.vddr S'.fls = S.fls S'.tipoH = T.tipo S'.dirH = T.dir S.tipo = S'.tipo S.dir = S'.dir
S' -> == T S'1	T.vddr = S'.vddr T.fls = S'.fls S'1.vddr = S'.vddr S'1.fls = S'.fls **Si** equivalentes(S'.tipoH, T.tipo) **Entonces** S'.dir = nuevaTemp() tipoTemp = **max**(S'.tipoH,T.tipo) d1 = **ampliar**(S'.dirH, S'.tipoH, tipoTemp) d2 = **ampliar**(T.dir, T.tipo, tipoTemp) genCod(S'.dir '=' d1.dir '==' d2.dir) genCod('if' S'.dir 'goto' T.vddr) genCod('goto' T.fls) S'.tipo = S'1.tipo S'1.tipoH = T.tipo S'1.dirH = T.dir **Sino** error("Tipos incompatibles") **FinSi**

S' -> != T S'1	T.vddr = S'.vddr
	T.fls = S'.fls
	S'1.vddr = S'.vddr
	S'1.fls = S'.fls
	Si equivalentes(S'.tipoH, T.tipo) **Entonces**
	  S'.dir = nuevaTemp()
	  tipoTemp = **max**(S'.tipoH,T.tipo)
	  d1 = **ampliar**(S'.dirH, S'.tipoH, tipoTemp)
	  d2 = **ampliar**(T.dir, T.tipo, tipoTemp)
	  genCod(S'.dir '=' d1.dir '!=' d2.dir)
	  genCod('if' S'.dir 'goto' T.vddr)
	  genCod('goto' T.fls)
	  S'.tipo = S'1.tipo
	  S'1.tipoH = T.tipo
	  S'1.dirH = T.dir
	Sino
	  error("Tipos incompatibles")
	FinSi

S' -> epsilon	S'.tipo = S'.tipoH S'.dir = S'.dirH
T -> U T'	T.tipo = T'.tipo T.dir = T'.dir T'.tipoH = U.tipo T'.dirH = U.dir T'.vddr = T.vddr T'.fls = T.fls

T' -> < U	**Si** equivalentes(T'.tipoH, U.tipo) **Entonces**
	  T'.tipo = **INT**
	  T'.dir = nuevaTemporal()
	  tipoTemp = **max**(T'.tipoH, U.tipo)
	  d1 = **ampliar**(T'.dirH, T'.tipoH, tipoTemp)
	  d2 = **ampliar**(U.dir, U.tipo, tipoTemp)
	  genCod(T'.dir '=' d1.dir '<' d2.dir)
	  genCod('if' T'.dir 'goto' T'.vddr)
	  genCod('goto' T'.fls)
	Sino
	  error("Tipos incompatibles")
	FinSi

T' -> <= U	**Si** equivalentes(T'.tipoH, U.tipo) **Entonces**
	  T'.tipo = **INT**
	  T'.dir = nuevaTemporal()
	  tipoTemp = **max**(T'.tipoH, U.tipo)
	  d1 = **ampliar**(T'.dirH, T'.tipoH, tipoTemp)
	  d2 = **ampliar**(U.dir, U.tipo, tipoTemp)
	  genCod(T'.dir '=' d1.dir '<=' d2.dir)
	  genCod('if' T'.dir 'goto' T'.vddr)
	  genCod('goto' T'.fls)
	Sino
	  error("Tipos incompatibles")
	FinSi

T' -> >= U	**Si** equivalentes(T'.tipoH, U.tipo) **Entonces**
	  T'.tipo = **INT**
	  T'.dir = nuevaTemporal()
	  tipoTemp = **max**(T'.tipoH, U.tipo)
	  d1 = **ampliar**(T'.dirH, T'.tipoH, tipoTemp)
	  d2 = **ampliar**(U.dir, U.tipo, tipoTemp)
	  genCod(T'.dir '=' d1.dir '>=' d2.dir)
	  genCod('if' T'.dir 'goto' T'.vddr)
	  genCod('goto' T'.fls)
	Sino
	  error("Tipos incompatibles")
	FinSi

T' -> > U	**Si** equivalentes(T'.tipoH, U.tipo) **Entonces**
	  T'.tipo = **INT**
	  T'.dir = nuevaTemporal()
	  tipoTemp = **max**(T'.tipoH, U.tipo)
	  d1 = **ampliar**(T'.dirH, T'.tipoH, tipoTemp)
	  d2 = **ampliar**(U.dir, U.tipo, tipoTemp)
	  genCod(T'.dir '=' d1.dir '>' d2.dir)
	  genCod('if' T'.dir 'goto' T'.vddr)
	  genCod('goto' T'.fls)
	Sino
	  error("Tipos incompatibles")
	FinSi

T' -> epsilon	T'.tipo = T'.tipoH T'.dir = T'.dirH
U -> V U'	U.tipo = U'.tipo U.dir = U'.dir U'.tipoH = V.tipo U'.dirH = V.dir

U' -> + V U'1	**Si** equivalentes(U'.tipoH, V.tipo) **Entonces**
	  U'.tipo = U'1.tipo
	  U'.dir = U'1.dir
	  U'1.tipoH = **max**(U'.tipoH, V.tipo)
	  U'1.dirH = nuevaTemporal
	  d1 = **max**(U'.dirH, U'.tipoH, U'1.tipoH)
	  d2 = **max**(V.dir, V.tipo, U'1.tipoH)
	  genCod(U'1.dirH '=' d1 '+' d2.dir)
	Sino
	  error("Tipos incompatibles")
	FinSi

U' -> - V U'1	**Si** equivalentes(U'.tipoH, V.tipo) **Entonces**
	  U'.tipo = U'1.tipo
	  U'.dir = U'1.dir
	  U'1.tipoH = **max**(U'.tipoH, V.tipo)
	  U'1.dirH = nuevaTemporal
	  d1 = **max**(U'.dirH, U'.tipoH, U'1.tipoH)
	  d2 = **max**(V.dir, V.tipo, U'1.tipoH)
	  genCod(U'1.dirH '=' d1 '-' d2.dir)
	Sino
	  error("Tipos incompatibles")
	FinSi

U' -> epsilon	U'.tipo = U'.tipoH U'.dir = U'.dirH
V -> W V'	V.tipo = V'.tipo V.dir = V'.dir V'.tipoH = W.tipo V'.dirH = W.dir

V' -> * W V'1	**Si** equivalentes(V'.tipoH, W.tipo) **Entonces**
	  V'.tipo = V'1.tipo
	  V'.dir = V'1.dir
	  V'1.tipoH = **max**(V'.tipoH, W.tipo)
	  V'1.dirH = nuevaTemporal()
	  d1 = **max**(V'.dirH, V'.tipoH, V'1.tipoH)
	  d2 = **max**(W.dir, W.tipo, V'1.tipoH)
	  genCod(V'1.dirH '=' d1 '*' d2.dir)
	Sino
	  error("Tipos incompatibles")
	FinSi

V' -> / W V'1	**Si** equivalentes(V'.tipoH, W.tipo) **Entonces**
	  V'.tipo = V'1.tipo
	  V'.dir = V'1.dir
	  V'1.tipoH = **max**(V'.tipoH, W.tipo)
	  V'1.dirH = nuevaTemporal()
	  d1 = **max**(V'.dirH, V'.tipoH, V'1.tipoH)
	  d2 = **max**(W.dir, W.tipo, V'1.tipoH)
	  genCod(V'1.dirH '=' d1 '/' d2.dir)
	Sino
	  error("Tipos incompatibles")
	FinSi

V' -> % W V'1	**Si** equivalentes(V'.tipoH, W.tipo) **Entonces**
	  V'.tipo = V'1.tipo
	  V'.dir = V'1.dir
	  V'1.tipoH = **INT**
	  V'1.dirH = nuevaTemporal()
	  genCod(V'1.dirH '=' V' '%' W.dir)
	Sino
	  error("Tipos incompatibles")
	FinSi

V' -> epsilon	V'.tipo = V'.tipoH V'.dir = V'.dirH
W -> ! W1	W.dir = nuevaTemporal() W.tipo = W1.tipo genCod(W.dir '=' '!' W1.dir)
W -> - W1	W.dir = nuevaTemporal() W.tipo = W1.tipo genCod(W.dir '=' '-' W1.dir)
W -> X	W.dir = X.dir W.tipo = X.tipo
X -> (Q)	X.dir = Q.dir X.tipo = Q.tipo
X -> **id** X'	X'.base = id.lexval X.dir = X'.dir X.tipo = X'.tipo
X -> **numero**	X.dir = numero.lexval X.tipo = numero.lextipo
X -> **cadena**	TablaCadenas.agregar(cadena.lexval) X.dir = TablaCadenas.getUltimaPos() X.tipo = cadena
X -> **true**	X.dir = 'true' X.tipo = int
X -> **false**	X.dir = 'false' X.tipo = int
X '-> AA	AA.base = X'.base X'.dir = nuevaTemp() X'.tipo = AA.tipo genCod(X'.dir '=' X'.base '['AA.dir']')

X' -> (Y)	**Si** PilaTS.fondo().buscar(X'.base) **Entonces**
	  Si PilaTS.fondo().getVar(X'.base) = '**func**' **Entonces**
	  Si equivalentesListas(PilaTS.fondo().getArgs(X'.base), Y.lista) **Entonces**
	  X'.tipo = PilaTS.top().getTipo(X'.base)
	  X'.dir = nuevaTemp()
	  genCode(X'.dir '=' 'call **X'.base** ',' Y.lista.tam)
	  Sino
	  error("El número o tipo de parámetros no coincide)
	  FinSi
	  Sino
	  error("El id no es una función")
	  FinSi
	Sino
	  error("El id no está declarado")

X' -> epsilon	X'.dir = X'.base X'.tipo = PilaTS.top.getTipo(X'.dir)
Y -> Z	Y.lista = Z.lista
Y -> epsilon	Y.lista = NULO
Z -> Q Z'	Z.lista = Z'.lista Z.lsita.agregar(Q.tipo) genCode('param' Q.dir)
Z' -> , Q Z'1	Z'.lista = Z'1.lista Z.lsita.agregar(Q.tipo) genCode('param' Q.dir)
Z' -> epsilon	Z'.lista = nuevaListaArgs()

AA -> [Q] AA'	**Si** PilaTS.top().buscar(AA.base) **Entonces**
	  Si Q.tipo = **INT Entonces**
	  tipoTmp = PilaTS.top().getTipo(Q.base)
	  Si PilaTT.top().getNombre(tipoTmp) = 'array' **Entonces**
	  AA'.tipo = PilaTT.top().getTipoBase(tipoTmp)
	  AA'.dir = nuevaTemporal()
	  AA'.dir = PilaTT.top().getTam(AA'.tipo)
	  genCod(AA.dir '=' Q.dir '*' AA.tam)
	  AA.dir = AA'.dirS
	  AA.tipo = AA'.tipoS
	  Sino
	  error("El id no es un arreglo")
	  FinSi
	  Sino
	  error("El índice del arreglo debe ser entero")
	  FinSi
	Sino
	  error("El id no está declarado")
	FinSi

AA' -> [Q] AA'1	**Si** Q.tipo = **INT Entonces**
	  Si PilaTT.top().getNombre(AA'.tipo) = 'array' **Entonces**
	  AA'1.tipo = PilaTT.top().getTipoBase(AA'.tipo)
	  dirTmp = nuevaTemporal()
	  AA'1.dir = nuevaTemporal()
	  AA'1.tam = PilaTT.top().getTam(AA'.tipo)
	  genCod(dirTmp '=' Q.dir '*' AA'1.tam)
	  genCod(AA'1.dir = '=' AA'.dir + dirTmp)
	  AA'.dir = AA'1.dirS
	  AA'.tipo = AA'1.tipoS
	  Sino
	  error("El id no es un arreglo")
	  FinSi
	Sino
	  error("El índice del arreglo debe ser un entero")
	FinSi

AA' -> epsilon	AA'.dirS=AA'.dir AA'.tipoS = AA'.tipo

Esquema de traducción

```
A -> { PilaTS.push(nuevaTablaTS())
       PilaTT.push(nuevaTablaTT())
       dir = 0 }
     B G

B -> C
     { F.tipo = C.tipo }
     F; B

B -> epsilon

C -> D
     { E.base = D.base }
     E
     { C.tipo = E.tipo }

D -> int
     { D.tipo = int }

D -> float
     {D.tipo = float}

D -> char
     {D.tipo = char}

D -> double
     {D.tipo = double}

D -> void
     {D.tipo = void}

E -> [numero]
     { E1.base = E.base }
     E1
     { E.tipo = PilaTT.top().insertar("array", num.val, E1.tipo) }

E -> epsilon
     { E.tipo = E.base }

F -> id
     { F'.tipo = F.tipo }
     F'
     { Si ! PilaTS.top().buscar(id) Entonces
         PilaTS.top().insertar(id, F.tipo, dir, "var",    NULO))
         dir = dir + PilaTT.top().getTam(C.tipo)
       SiNo
         error ("El id no está declarado")
       FinSi }

F' -> , id
     { F'1.tipo = F'.tipo }
     F'1
     { Si ! PilaTS.top().buscar(id) Entonces
         PilaTS.top().insertar(id, F'.tipo, dir, "var",    NULO))
         dir = dir + PilaTT.top().getTam(C.tipo)
       SiNo
         error ("El id no está declarado")
       FinSi }

F' -> epsilon
```

G -> func C id (H) { ListaRetorno = NULL PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos) PilaDir.push(dir) dir = 0 Si ! PilaTS.top().buscar(id) Entonces Si equivalentesLista(ListaRetorno, C.tipo) Ent PilaTS.top().insertar(id,C.tipo,-,`func` H.lista) genCod(label(id)) J.sig = nuevaEtq() } J G { genCod(label(J.sig)) Sino error("Retorno no coincide") FinSi Sino error("ID no declarado") FinSi PilaTS.pop() PilaTT.pop() dir = PilaDir.pop() }							
G -> epsilon							
H -> I {H.lista = I.lista}							
H -> epsilon {H.lista = NULL}							
I -> C id I' { Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, C.tipo, dir, "param",NULL) dir = dir + PilaTT.top().getTam(F.tipo) Sino error ("ID no está declarado") FinSi I.lista = I'.lista I.lista.agregar(C.tipo) }							
I' -> , C id { Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, C.tipo, dir, "param",NULL) dir = dir + PilaTT.top().getTam(C.tipo) Sino error ("ID no está declarado") FinSi } I'1 { I'.lista = I'1.lista I.lista.agregar(C.tipo) }							
I' -> epsilon { I'.lista = nuevaListaArgs() I'.lista.agregar(C.tipo) }							
J -> { B {K.sig = J.sig} K } { genCod(label(K.sig)) }							
K -> { L.sig = K'.sig } L { K'.sigH = K.sig } K'							
K' -> { L.sig = K'1.sig } L { K'1.sigH = K'.sigH } K'1 { K'.sig = nuevaEtq() genCod(label(K'.sig)) }							
K' -> epsilon { K'.sig = K'.sigH }							

L -> P = Q ; { Si equivalentes(P.tipo, Q.tipo) Entonces d1 = reducir(Q.dir, Q.tipo, P.tipo) genCod(P.dir `=` d1) Sino error("Tipos incompatibles") FinSi }							
L -> if({ Q.vddr = nuevaEtq() Q.fls = nuevaEtq() } **Q)** { L1.sig = L.sig } **L1** { genCod(label(Q.vddr)) }							
L -> if({ Q.vddr = nuevaEtq() Q.fls = nuevaEtq() } **Q)** { L1.sig = L.sig } **L1 else** { L2.sig = L.sig } **L2** { genCod(label(Q.vddr)) genCod(`goto` L.sig) genCod(label(Q.fls)) }							
L -> while({ Q.vddr = nuevaEtq() Q.fls = L.sig } **Q)** { L1.sig = nuevaEtq() } **L1** { genCod(label(L1.sig)) genCod(label(Q.vddr)) genCod(`goto` L1.sig) }							
L -> do { L1.sig = nuevaEtq() } **L1 while(** { Q.vddr = nuevaEtq() Q.fls = L.sig } **Q)** { genCod(label(L1.sig)) genCod(label(Q.vddr)) }							
L -> break; { genCod(goto L.sig) }							
L -> { J.sig = L.sig } **J**							
L -> return U; { ListaRetorno.agregar(U.tipo) genCod(`return` U.dir) }							
L -> return; { ListaRetorno.agregar(void) genCod(`return`) }							
L -> switch(Q) { { M.etqPrueba = nuevaEtq() genCode(`goto` M.etqPrueba) M.sig = L.sig M.id = Q.dir } **M }** { genCode(label(M.etqPrueba)) genCode(M.prueba) }							
L -> print U; { genCod(`print` U.val) }							
L -> scan P { genCod(`scan` P.dir) }							

```
M -> { N.sig = M.sig }
       N
       { M1.sig = M.sig }
       M1
       { M.prueba = M.prueba || M1.prueba }
M -> { O.sig = M.sig }
       O
       { M.prueba = O.prueba }
M -> epsilon
       { M.prueba = '' }
N -> case num:
       { K.sig = M.sig }
       K
       { N.inicio = nuevaEtq()
         N.prueba = genCod(if N.id `==` num.lexval `goto` M.inicio)
         genCode(label(N.inicio)) }
O -> default:
       { K.sig = O.sig }
       K
       { O.inicio = nuevaEtq()
         O.prueba = genCod(`goto` O.inicio)
         genCod(label(O.inicio)) }
P -> id P'
       { P.base = id.lexval
         P.dir = P'.dir
         P.tipo = P'.tipo }
P' -> { AA.base = P'.base }
       AA
       { P'.dir = AA.dir
         P'.tipo = AA.tipo }
P' -> epsilon
       { Si PilaTS.top().buscar(P'.base) Entonces
            P'.dir = P'.base
            P'.tipo = PilaTS.top().getTipo(P'.dir)
         Sino
            error("El id no está declarado")
         FinSi }
Q -> { R.vddr = Q.vddr
       R.fls = nuevoIndice() }
     R
     { Q'.tipoH = R.tipo
       Q'.lista_indices = nuevaListaIndices()
       Q'.lista_indices.agregar(R.fls) }
     Q'
     { Q.tipo = Q'.tipoS
       genCod(label(R.fls)) }
Q' -> ||
       { Si equivalentes(Q'.tipoH, R.tipo) Entonces
                R.vddr = Q.vddr
                R.fls = nuevoIndice() }
           R
           {    Q'1.tipoH = R.tipo
                Q'1.vddr = Q.vddr
                Q'1.fls = Q.fls
                Q'1.lista_indices = Q'1.lista_indices
                Q'1.lista_indices.agregar(R.fls) }
           Q'1
           {    Q'.tipoS = Q'1.tipoS
                genCod(label(Q1.fls))
             Sino
                error("Tipos incompatibles")
         FinSi}
Q' -> epsilon
       { reemplazarIndices(Q'.lista_indices, Q'.fls, cuadruplas)
         Q'.tipoS = int }
```

R -> { S.vddr = nuevoIndice() S.fls = R.fls } S { R'.tipoH = S.tipo R'.lista_indices = nuevaListaIndices() R'.lista_indices.agregar(S.vddr) } R' { R.tipo = R'.tipoS genCod(label(S.vddr)) }							
R' -> && { Si equivalentes(R'.tipoH, S.tipo) Entonces S.vddr = nuevoIndice() S.fls = R'.fls (aunque dice R, no R') } S { R'1.tipoH = R'.tipo (lo mismo) R'1.vddr = bool.vddr R'1.fls = bool.fls R'1.lista_indices = bool'1.lista_indices R'1.lista_indices.agregar(S.vddr) } R'1 { R'.tipoS = R'1.tipoS genCod(label(S.vddr)) Sino error("Tipos incompatibles") FinSi }							
R' -> epsilon { reemplazarIndices(R'.lista_indices, R'.vddr, cuadruplas) R'.tipoS = INT }							
S -> { T.vddr = S.vddr T.fls = S.fls } T { S'.vddr = S.vddr S'.fls = S.fls S'.tipoH = T.tipo S'.dirH = T.dir } S' { S.tipo = S'.tipo S.dir = S'.dir }							
S' -> == { T.vddr = S'.vddr T.fls = S'.fls } T { S'1.vddr = S'.vddr S'1.fls = S'.fls } S'1 { Si equivalentes(S'.tipoH, T.tipo) Entonces S'.dir = nuevaTemp() tipoTemp = max(S'.tipoH,T.tipo) d1 = ampliar(S'.dirH, S'.tipoH, tipoTemp) d2 = ampliar(T.dir, T.tipo, tipoTemp) genCod(S'.dir '=' d1.dir '==' d2.dir) genCod('if' S'.dir 'goto' T.vddr) genCod('goto' T.fls) S'.tipo = S'1.tipo S'1.tipoH = T.tipo S'1.dirH = T.dir Sino error("Tipos incompatibles") FinSi }							

S' -> != { T.vddr = S'.vddr T.fls = S'.fls } T { S'1.vddr = S'.vddr S'1.fls = S'.fls } S'1 { Si equivalentes(S'.tipoH, T.tipo) Entonces S'.dir = nuevaTemp() tipoTemp = max(S'.tipoH,T.tipo) d1 = ampliar(S'.dirH, S'.tipoH, tipoTemp) d2 = ampliar(T.dir, T.tipo, tipoTemp) genCod(S'.dir '=' d1.dir '!=' d2.dir) genCod('if' S'.dir 'goto' T.vddr) genCod('goto' T.fls) S'.tipo = S'1.tipo S'1.tipoH = T.tipo S'1.dirH = T.dir Sino error("Tipos incompatibles") FinSi }							
S' -> epsilon { S'.tipo = S'.tipoH S'.dir = S'.dirH }							
T -> U { T'.tipoH = U.tipo T'.dirH = U.dir T'.vddr = T.vddr T'.fls = T.fls } T' { T.tipo = T'.tipo T.dir = T'.dir }							
T'-> < U { Si equivalentes(T'.tipoH, U.tipo) Entonces T'.tipo = INT T'.dir = nuevaTemporal() tipoTemp = max(T'.tipoH, U.tipo) d1 = ampliar(T'.dirH, T'.tipoH, tipoTemp) d2 = ampliar(U.dir, U.tipo, tipoTemp) genCod(T'.dir '=' d1.dir '<' d2.dir) genCod('if' T'.dir 'goto' T'.vddr) genCod('goto' T'.fls) Sino error("Tipos incompatibles") FinSi }							
T'-> <= U { Si equivalentes(T'.tipoH, U.tipo) Entonces T'.tipo = INT T'.dir = nuevaTemporal() tipoTemp = max(T'.tipoH, U.tipo) d1 = ampliar(T'.dirH, T'.tipoH, tipoTemp) d2 = ampliar(U.dir, U.tipo, tipoTemp) genCod(T'.dir '=' d1.dir '<=' d2.dir) genCod('if' T'.dir 'goto' T'.vddr) genCod('goto' T'.fls) Sino error("Tipos incompatibles") FinSi }							

T' -> >= U
{ Si equivalentes(T'.tipoH, U.tipo) Entonces
T'.tipo = INT
T'.dir = nuevaTemporal()
tipoTemp = max(T'.tipoH, U.tipo)
d1 = ampliar(T'.dirH, T'.tipoH, tipoTemp)
d2 = ampliar(U.dir, U.tipo, tipoTemp)
genCod(T'.dir '=' d1.dir '>=' d2.dir)
genCod('if' T'.dir 'goto' T'.vddr)
genCod('goto' T'.fls)
Sino
error("Tipos incompatibles")
FinSi }

T' -> > U
{ Si equivalentes(T'.tipoH, U.tipo) Entonces
T'.tipo = INT
T'.dir = nuevaTemporal()
tipoTemp = max(T'.tipoH, U.tipo)
d1 = ampliar(T'.dirH, T'.tipoH, tipoTemp)
d2 = ampliar(U.dir, U.tipo, tipoTemp)
genCod(T'.dir '=' d1.dir '>' d2.dir)
genCod('if' T'.dir 'goto' T'.vddr)
genCod('goto' T'.fls)
Sino
error("Tipos incompatibles")
FinSi }

T' -> epsilon
{ T'.tipo = T'.tipoH
T'.dir = T'.dirH }

U -> V
{ U'.tipoH = V.tipo
U'.dirH = V.dir }
U'
{ U.tipo = U'.tipo
U.dir = U'.dir }

U' -> + V U'1
{ Si equivalentes(U'.tipoH, V.tipo) Entonces
U'.tipo = U'1.tipo
U'.dir = U'1.dir
U'1.tipoH = max(U'.tipoH, V.tipo)
U'1.dirH = nuevaTemporal
d1 = max(U'.dirH, U'.tipoH, U'1.tipoH)
d2 = max(V.dir, V.tipo, U'1.tipoH)
genCod(U'1.dirH '=' d1 '+' d2.dir)
Sino
error("Tipos incompatibles")
FinSi }

U' -> - V U'1
{ Si equivalentes(U'.tipoH, V.tipo) Entonces
U'.tipo = U'1.tipo
U'.dir = U'1.dir
U'1.tipoH = max(U'.tipoH, V.tipo)
U'1.dirH = nuevaTemporal
d1 = max(U'.dirH, U'.tipoH, U'1.tipoH)
d2 = max(V.dir, V.tipo, U'1.tipoH)
genCod(U'1.dirH '=' d1 '-' d2.dir)
Sino
error("Tipos incompatibles")
FinSi }

U' -> epsilon
{ U'.tipo = U'.tipoH
U'.dir = U'.dirH }

V -> W
{ V'.tipoH = W.tipo
V'.dirH = W.dir }
V'
{ V.tipo = V'.tipo
V.dir = V'.dir }

V' -> * W V'1
{ Si equivalentes(V'.tipoH, W.tipo) Entonces
V'.tipo = V'1.tipo
V'.dir = V'1.dir
V'1.tipoH = max(V'.tipoH, W.tipo)
V'1.dirH = nuevaTemporal()
d1 = max(V'.dirH, V'.tipoH, V'1.tipoH)
d2 = max(W.dir, W.tipo, V'1.tipoH)
genCod(V'1.dirH '=' d1 '*' d2.dir)
Sino
error("Tipos incompatibles")
FinSi }

V' -> / W V'1
{ Si equivalentes(V'.tipoH, W.tipo) Entonces
V'.tipo = V'1.tipo
V'.dir = V'1.dir
V'1.tipoH = max(V'.tipoH, W.tipo)
V'1.dirH = nuevaTemporal()
d1 = max(V'.dirH, V'.tipoH, V'1.tipoH)
d2 = max(W.dir, W.tipo, V'1.tipoH)
genCod(V'1.dirH '=' d1 '/' d2.dir)
Sino
error("Tipos incompatibles")
FinSi }

V' -> % W V'1
{ Si equivalentes(V'.tipoH, W.tipo) Entonces
V'.tipo = V'1.tipo
V'.dir = V'1.dir
V'1.tipoH = INT
V'1.dirH = nuevaTemporal()
genCod(V'1.dirH '=' V' '%' W.dir)
Sino
error("Tipos incompatibles")
FinSi }

V' -> epsilon
{ V'.tipo = V'.tipoH
V'.dir = V'.dirH }

W -> ! W1
{ W.dir = nuevaTemporal()
W.tipo = W1.tipo
genCod(W.dir '=' '!' W1.dir) }

W -> - W1
{ W.dir = nuevaTemporal()
W.tipo = W1.tipo
genCod(W.dir '=' '-' W1.dir) }

W -> X
{ W.dir = X.dir
W.tipo = X.tipo }

X -> (Q)
{ X.dir = Q.dir
X.tipo = Q.tipo }

X -> id
{ X'.base = id.lexval }
X'
{ X.dir = X'.dir
X.tipo = X'.tipo }

X -> numero
{ X.dir = numero.lexval
X.tipo = numero.lextipo }

X -> cadena
{ TablaCadenas.agregar(cadena.lexval)
X.dir = TablaCadenas.getUltimaPos()
X.tipo = cadena }

X -> true
{ X.dir = 'true'
X.tipo = int }

```
X -> false
       { X.dir = 'false'
         X.tipo = int }
```

```
X '-> { AA.base = X'.base }
        AA
        { X'.dir = nuevaTemp()
          X'.tipo = AA.tipo
          genCod(X'.dir '=' X'.base '['AA.dir']') }
```

```
X'-> (Y)
       { Si PilaTS.fondo().buscar(X'.base) Entonces
           Si PilaTS.fondo().getVar(X'.base) = 'func' Entonces
              Si equivalentesListas(PilaTS.fondo().getArgs(X'.base), Y.lista) Entonces
                 X'.tipo = PilaTS.top().getTipo(X'.base)
                 X'.dir = nuevaTemp()
                 genCode(X'.dir '=' 'call X'.base ',' Y.lista.tam)
              Sino
                 error(''El número o tipo de parámetros no coincide)
              FinSi
           Sino
              error(''El id no es una función'')
           FinSi
        Sino
             error(''El id no está declarado'')
        FinSi }
```

```
X' -> epsilon
       { X'.dir = X'.base
         X'.tipo = PilaTS.top.getTipo(X'.dir) }
```

```
Y -> Z
      { Y.lista = Z.lista }
```

```
Y -> epsilon
      { Y.lista = NULO }
```

```
Z -> Q Z'
      { Z.lista = Z'.lista
        Z.lista.agregar(Q.tipo)
        genCode('param' Q.dir) }
```

```
Z' -> , Q Z'
       { Z'.lista = Z'1.lista
         Z.lista.agregar(Q.tipo)
         genCode('param' Q.dir) }
```

```
Z' -> epsilon
       { Z'.lista = nuevaListaArgs() }
```

```
AA -> [Q]
       { Si PilaTS.top().buscar(AA.base) Entonces
            Si  Q.tipo = INT Entonces
               tipoTmp = PilaTS.top().getTipo(Q.base)
               Si PilaTT.top().getNombre(tipoTmp) = 'array' Entonces
                  AA'.tipo = PilaTT.top().getTipoBase(tipoTmp)
                  AA'.dir = nuevaTemporal()
                  AA'.dir = PilaTT.top().getTam(AA'.tipo)
                  genCod(AA.dir '=' Q.dir '*' AA.tam) }
       AA'
       {          AA.dir = AA'.dirS
                  AA.tipo = AA'.tipoS
             Sino
                error(''El id no es un arreglo'')
             FinSi
          Sino
             error(''El índice del arreglo debe ser entero'')
          FinSi
       Sino
          error(''El id no está declarado'')
       FinSi }
```

AA' -> [Q] { Si Q.tipo = INT Entonces Si PilaTT.top().getNombre(AA'.tipo) = 'array' Entonces AA'1.tipo = PilaTT.top().getTipoBase(AA'.tipo) dirTmp = nuevaTemporal() AA'1.dir = nuevaTemporal() AA'1.tam = PilaTT.top().getTam(AA'.tipo) genCod(dirTmp '=' Q.dir '*' AA'1.tam) genCod(AA'1.dir = '=' AA'.dir + dirTmp) } AA'1 { AA'.dir = AA'1.dirS AA'.tipo = AA'1.tipoS Sino error("El id no es un arreglo") FinSi Sino error("El índice del arreglo debe ser un entero") FinSi }							
AA' -> epsilon { AA'.dirS=AA'.dir AA'.tipoS = AA'.tipo }							