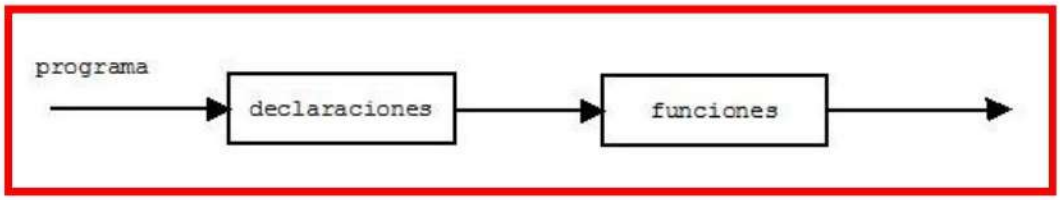
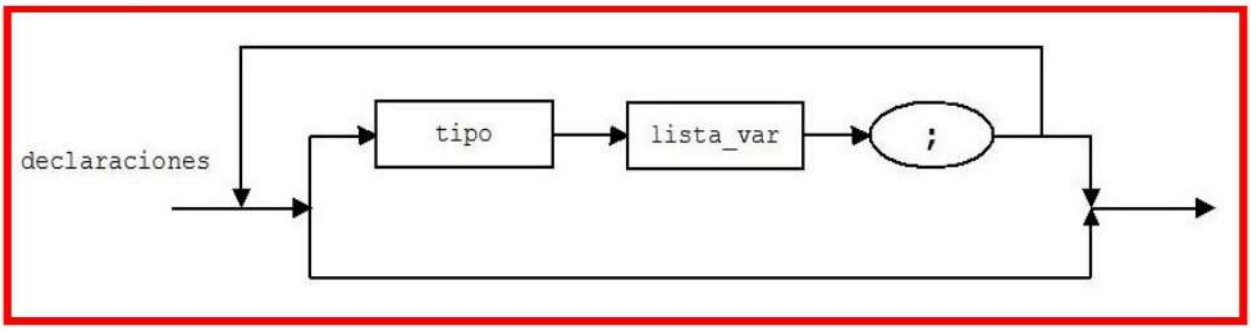
Diagramas de Sintaxis de la gramática.

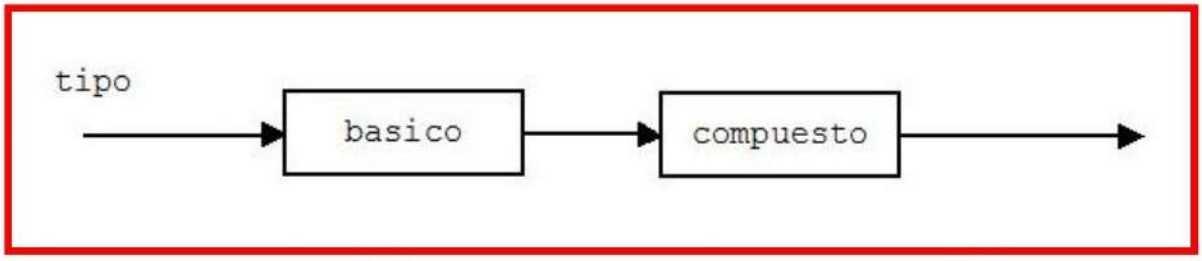
1) programa → declaraciones funciones



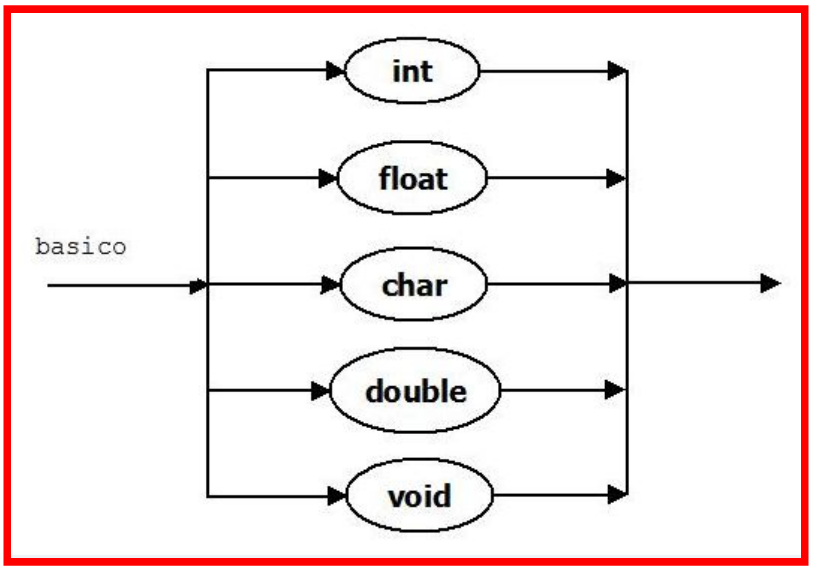
2) declaraciones → tipo lista var ; declaraciones | ε



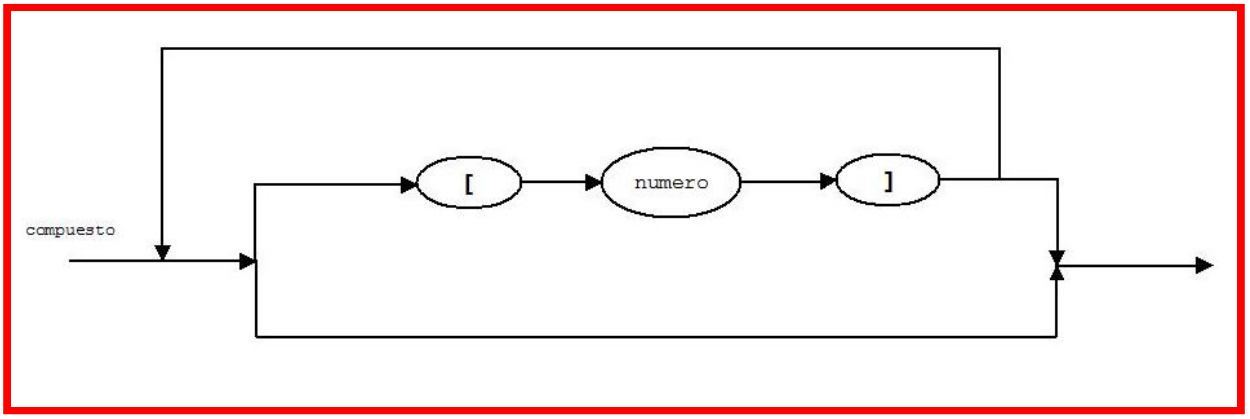
3) tipo → basico compuesto



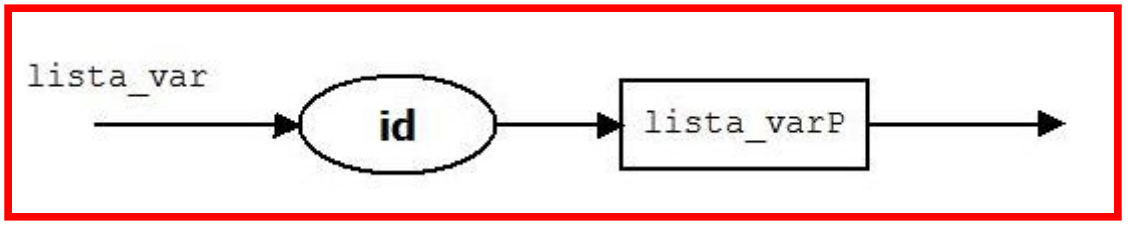
4) basico → **int** / **float** / **char** / **double**/ **void**



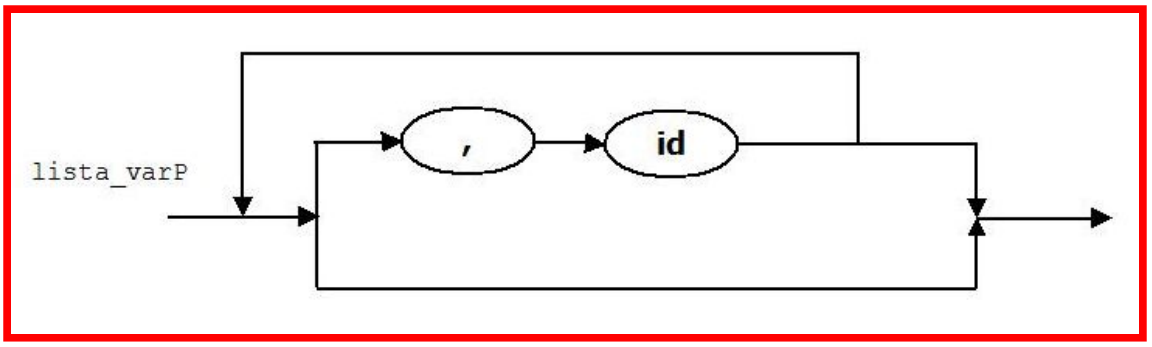
5) compuesto → **[numero]** compuesto | *ε*



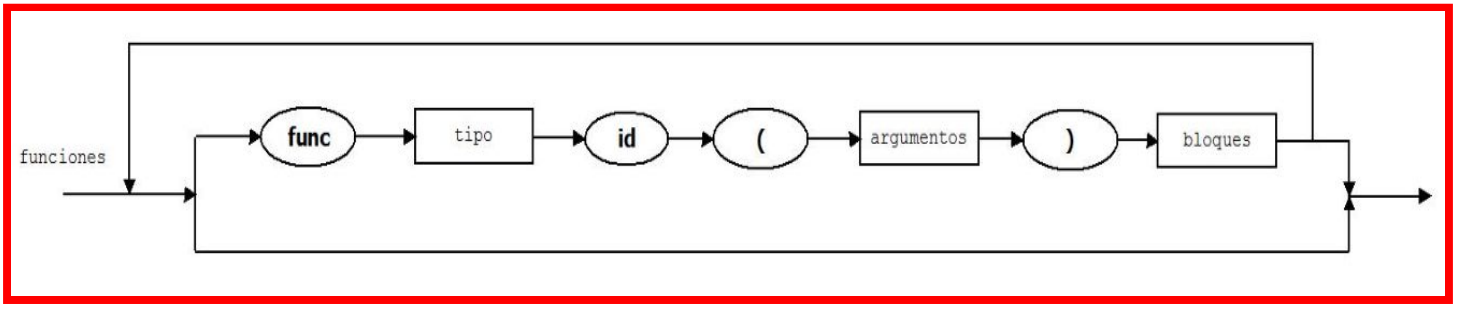
6) lista var → **id** lista_varP



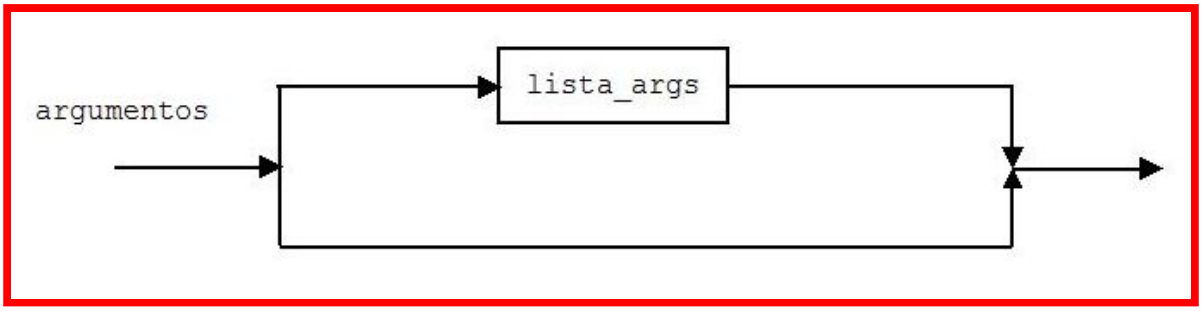
7) lista_varP → **,** **id** lista_varP **|** ε



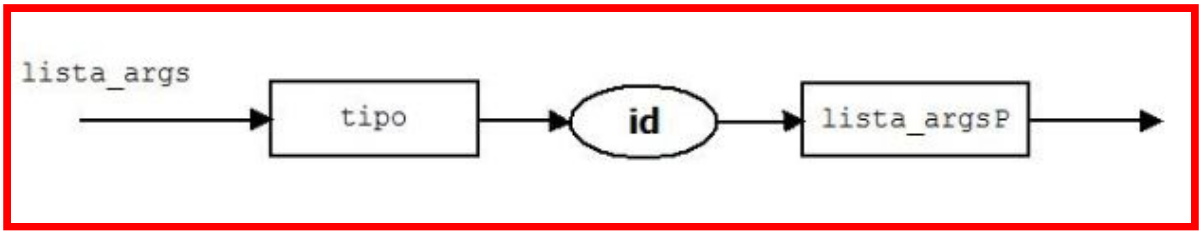
8) funciones → **func** tipo **id** **(** argumentos **)** bloque funciones | ε



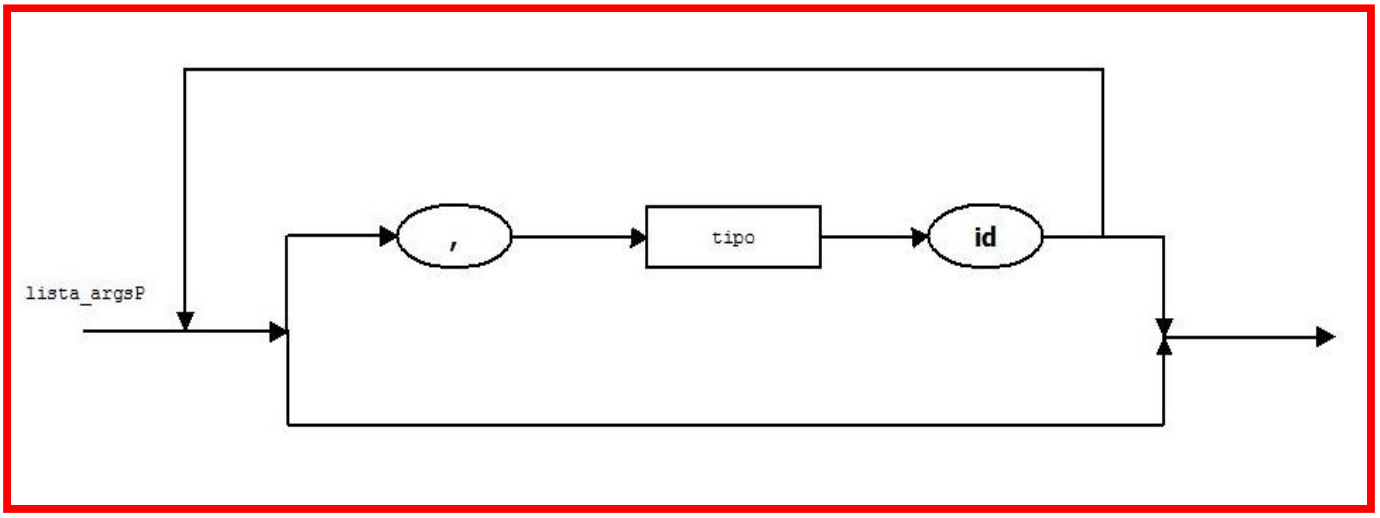
9) argumentos → lista_args | ε



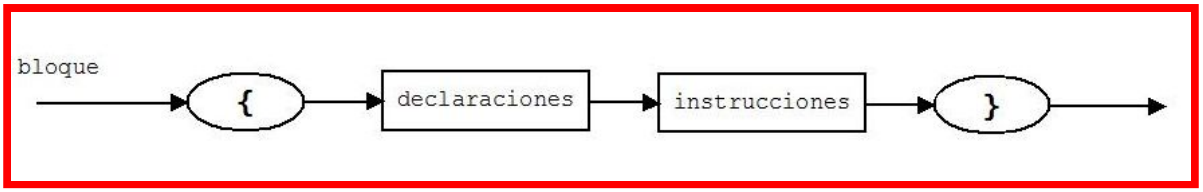
10) lista_args → tipo **id** lista_argsP



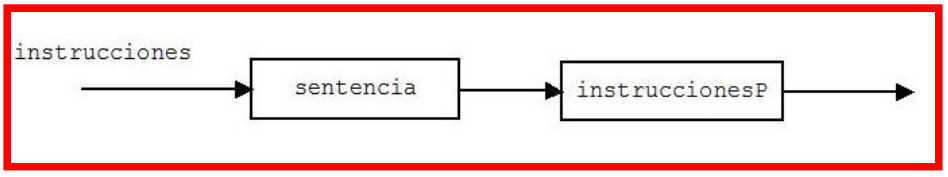
11) lista_argsP → **,** tipo **id** lista_argsP | *ε*



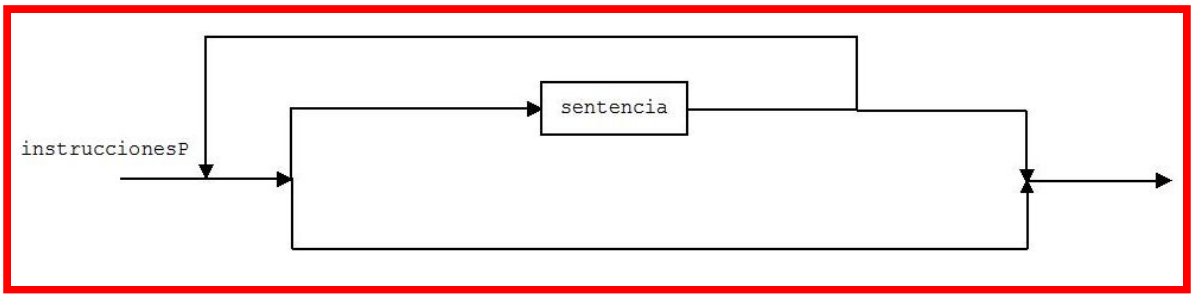
12) bloque → *{* declaraciones instrucciones *}*



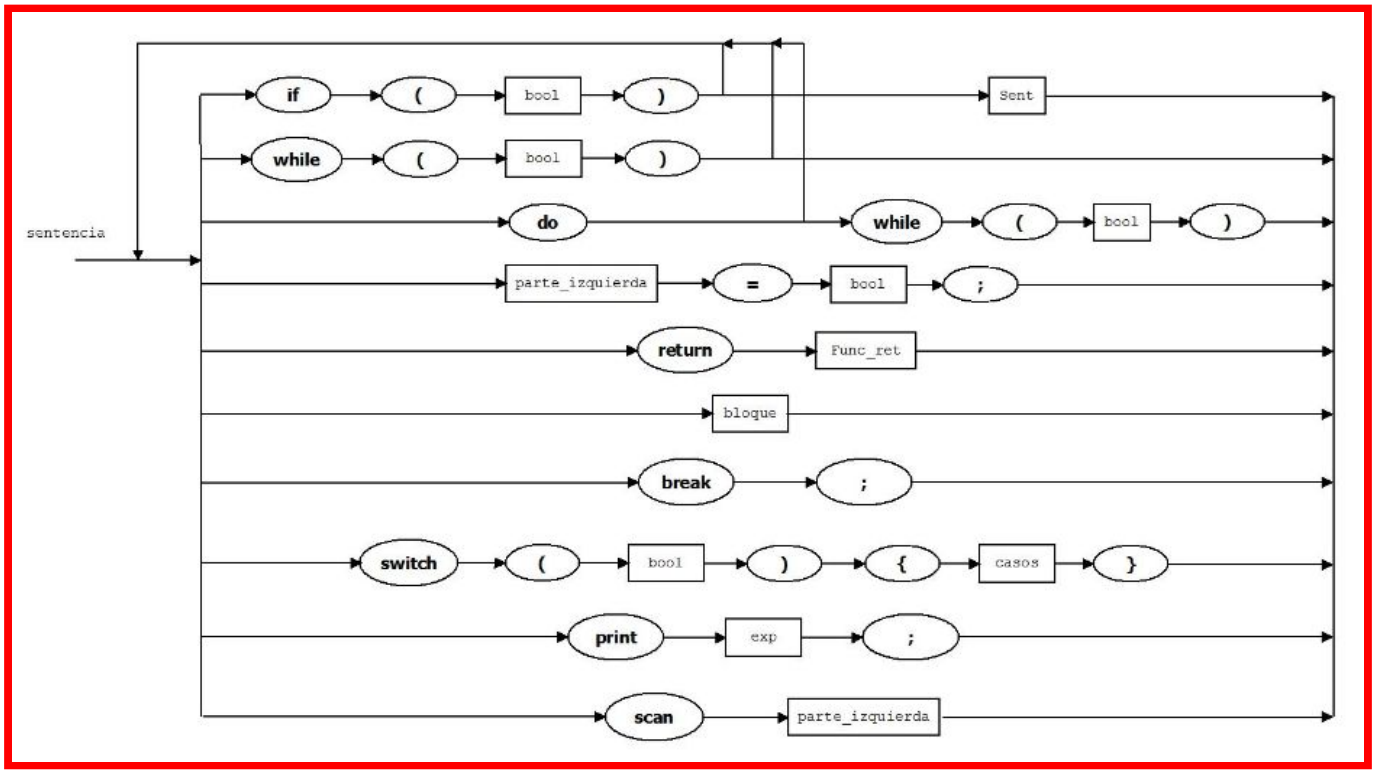
13) instrucciones → sentencia instruccionesP



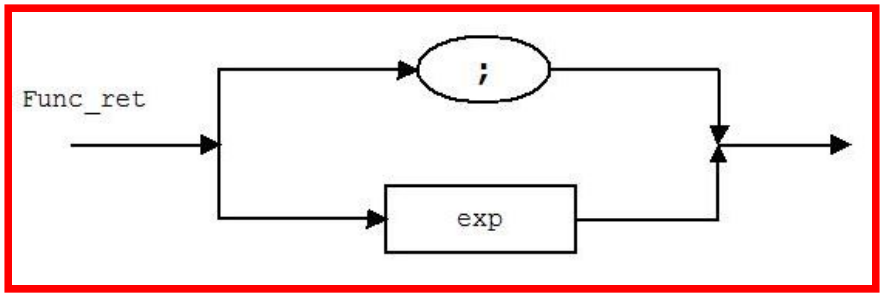
14) instruccionesP → sentencia instruccionesP| *ε*



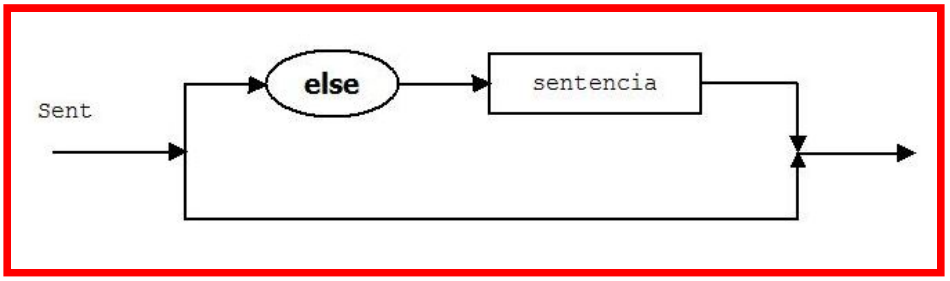
15) sentencia → parte_izquierda **=** bool**;** | **if(** bool **)** sentencia Sent | **while(** bool **)** sentencia | **do** sentencia **while(** bool **)** | **break ; | return** Func_ret | bloque | **switch(** bool **)** { casos } | **print** exp **;** | **scan** parte_izquierda



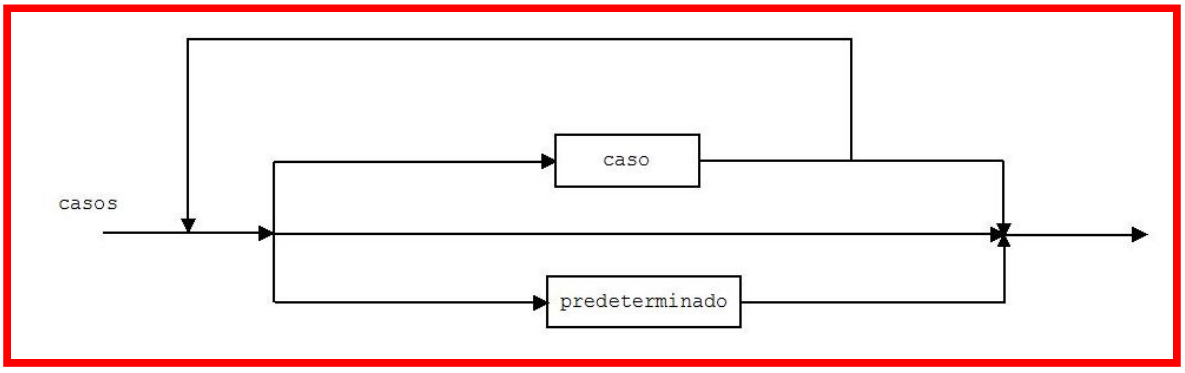
16) Func_ret → **;** | exp



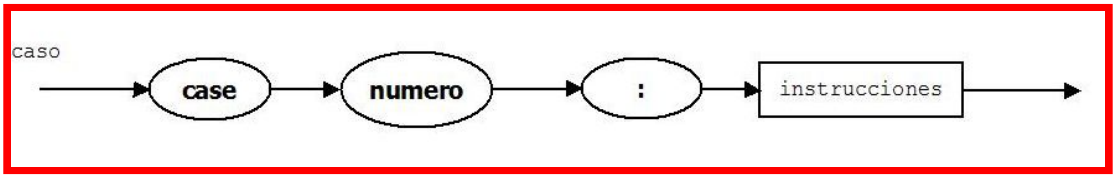
17) Sent → **else** sentencia | *ε*



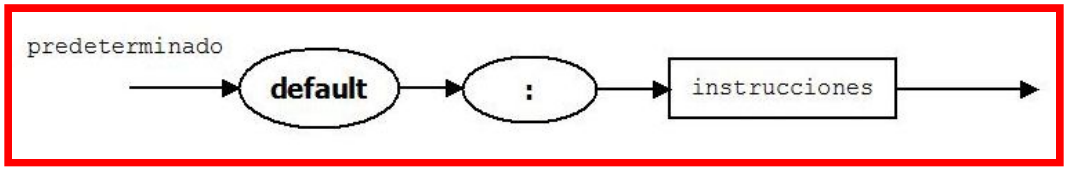
18) casos → caso casos | ε | predeterminado



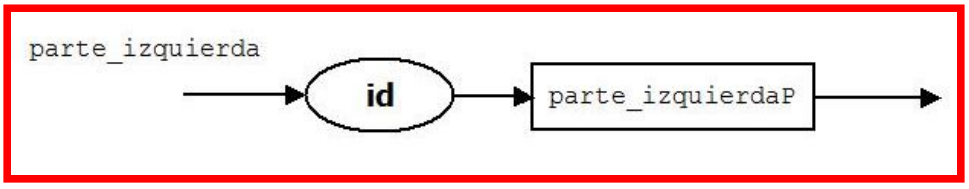
19) caso → **case numero:** instrucciones



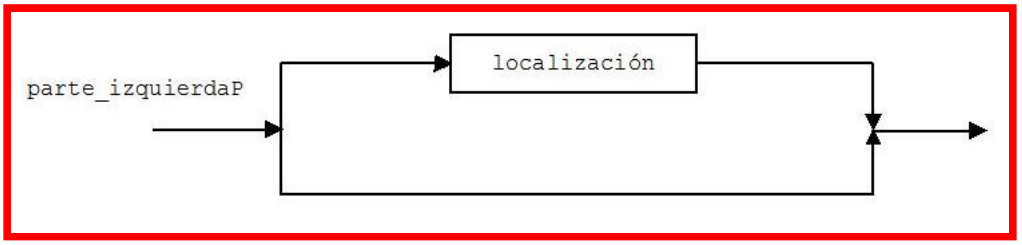
20) predeterminado → **default :** instrucciones



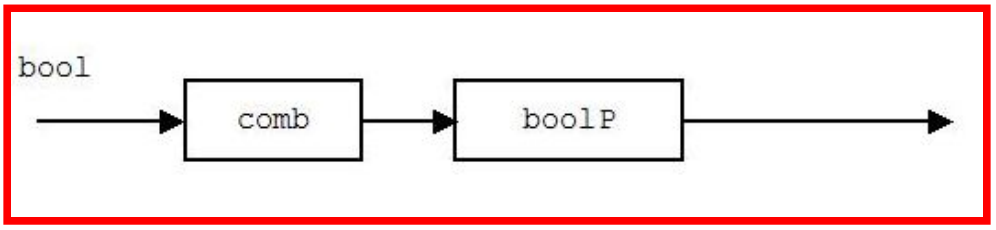
21) parte_izquierda → **id** parte_izquierdaP



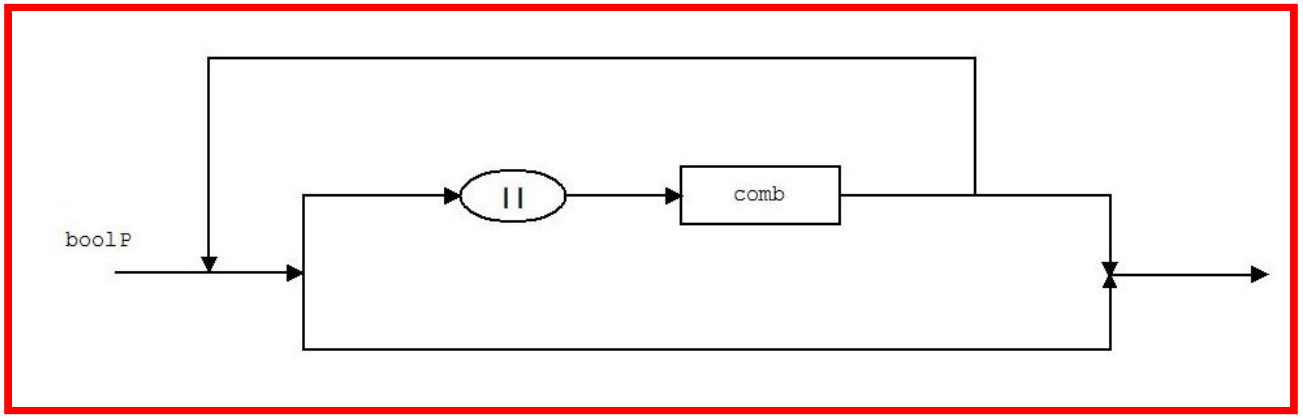
22) parte_izquierdaP → localización | ε



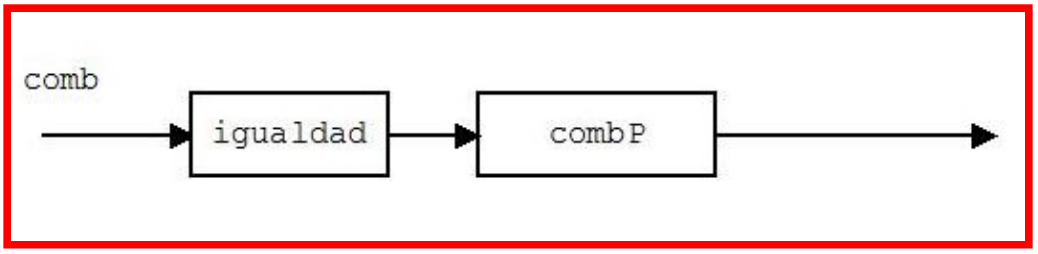
23) bool → comb boolP



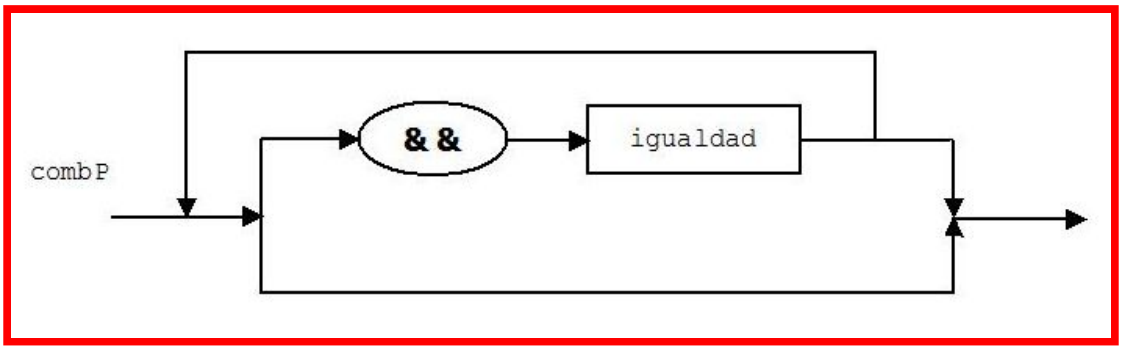
24) boolP → **||** comb boolP | *ε*



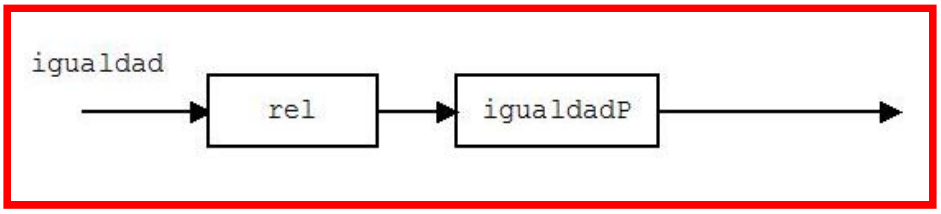
25) comb → igualdad combP



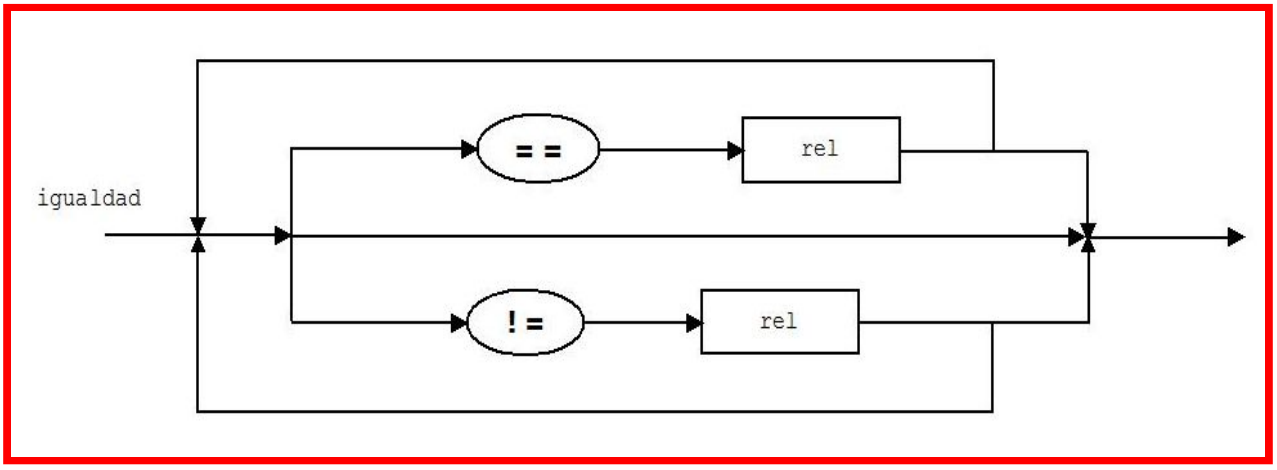
26) combP → **&&** igualdad combP | *ε*



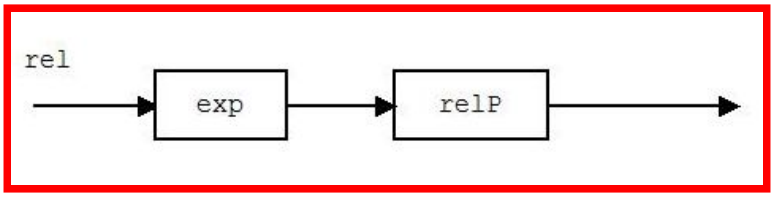
27) igualdad → rel igualdadP



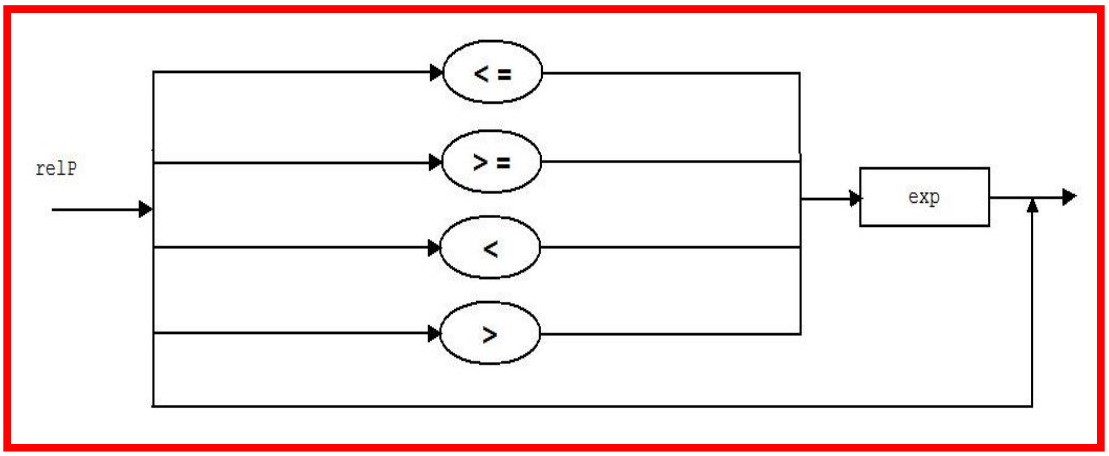
28) igualdadP → **==** rel igualdadP| **!=** rel igualdadP | ε



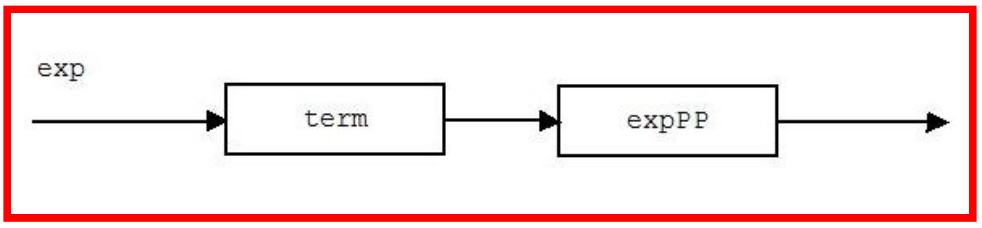
29) rel →exp relP



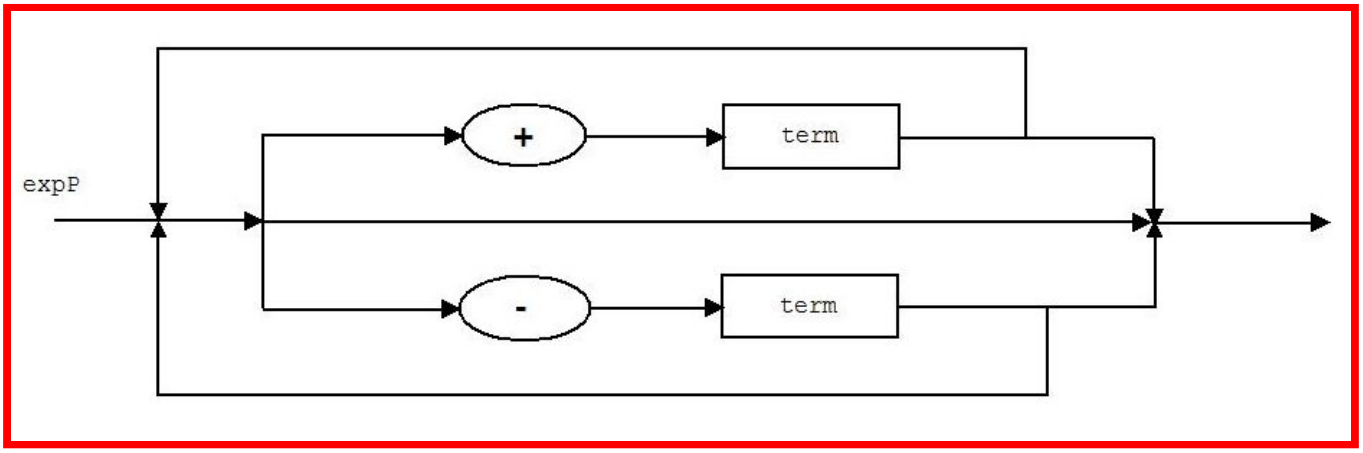
30) relP → **<=** exp | **>=** exp | **<** exp | **>** exp | ε



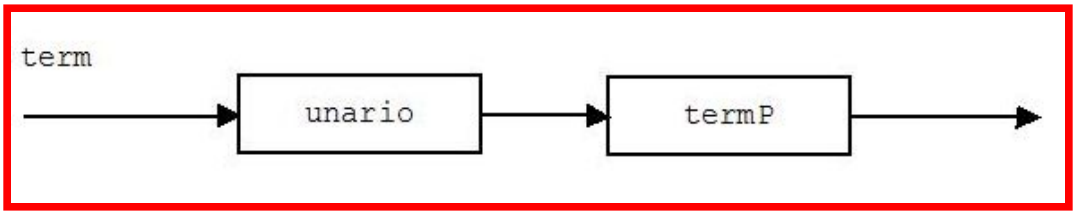
31) exp ⟶ term expP



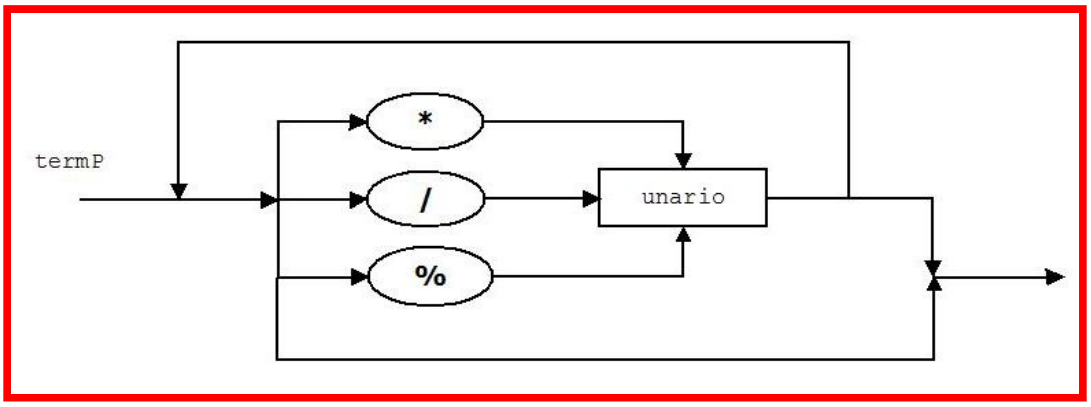
32) expP ⟶ **+** term expP / **−** term expP | ε



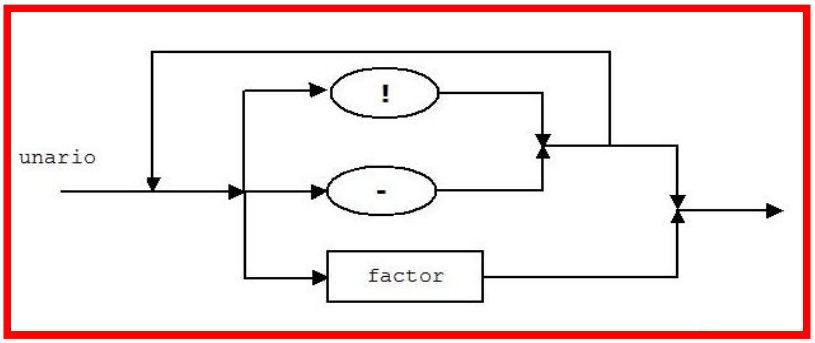
33) term ⟶ unario termP



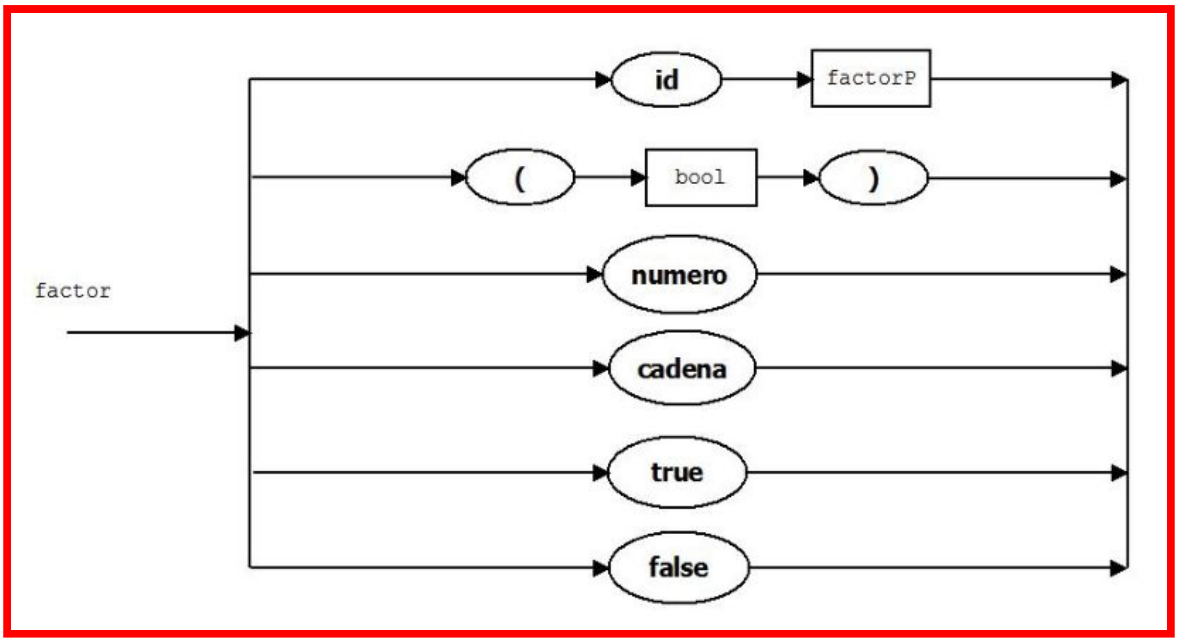
34) termP ⟶ ∗ unario termP / **/** unario termP / **%** unario termP / ε



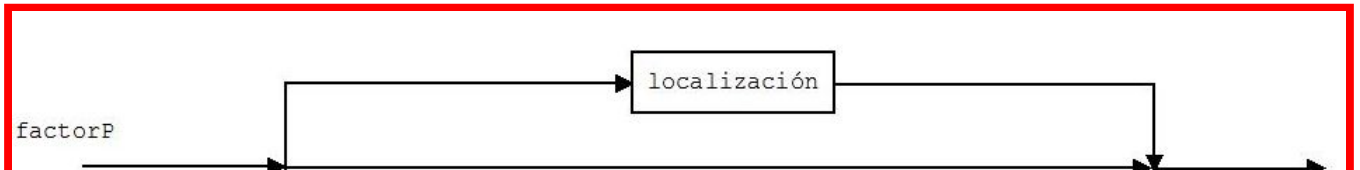
35) unario → **!**unario / **–** unario / factor



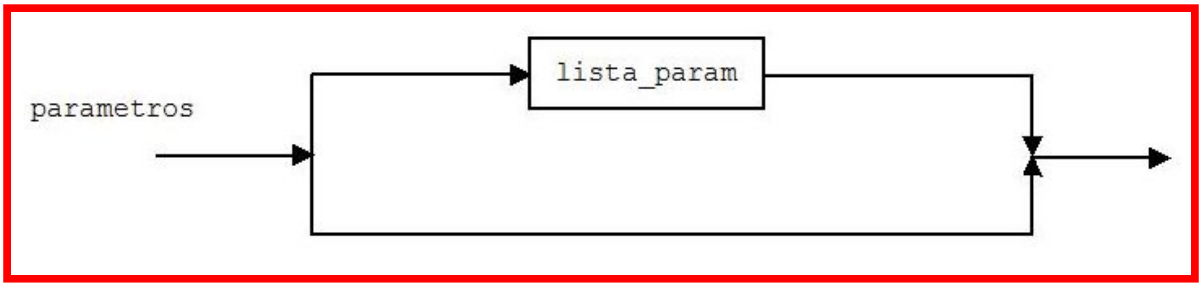
36) factor → **(**bool**)** / **numero** / **cadena** / **true** / **false** | **id** factorP



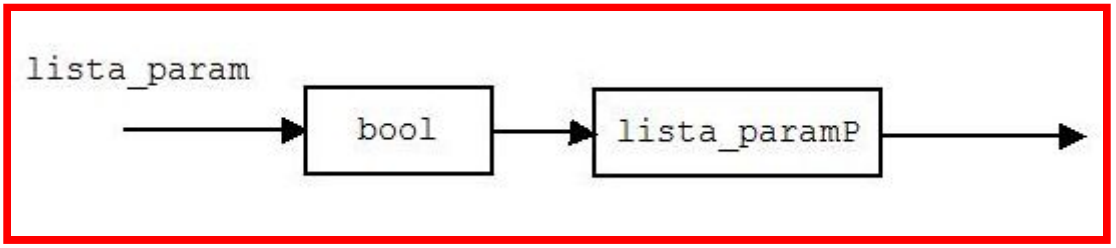
37) factorP → localización | **(**parametros**)** | *ε*



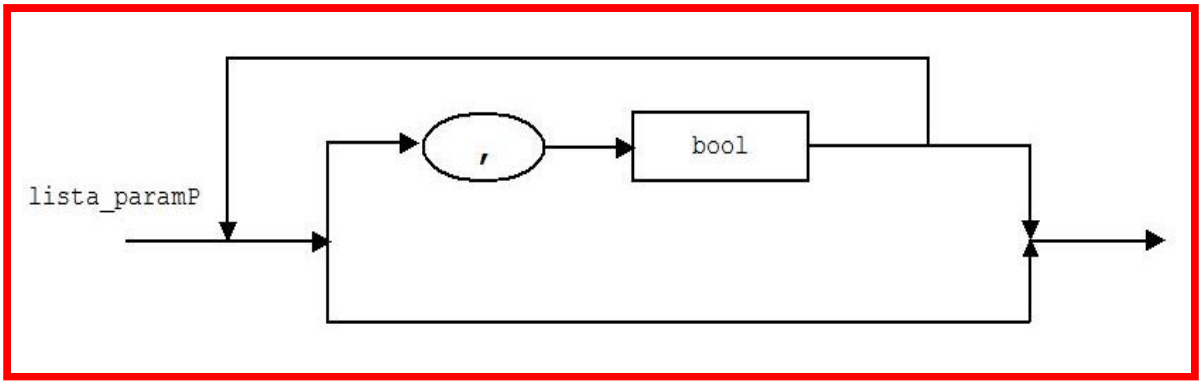
38) parametros → lista_param | ε



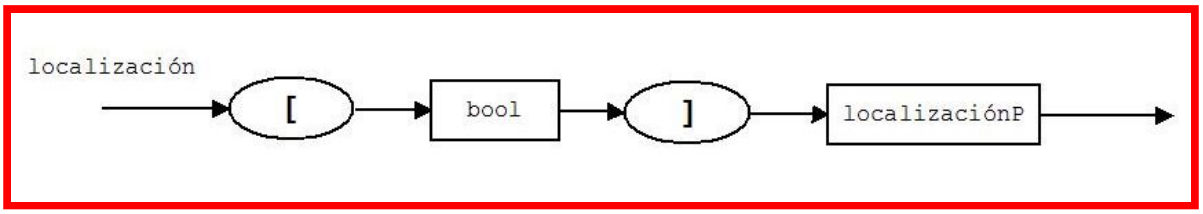
39) lista_param → bool lista_paramP



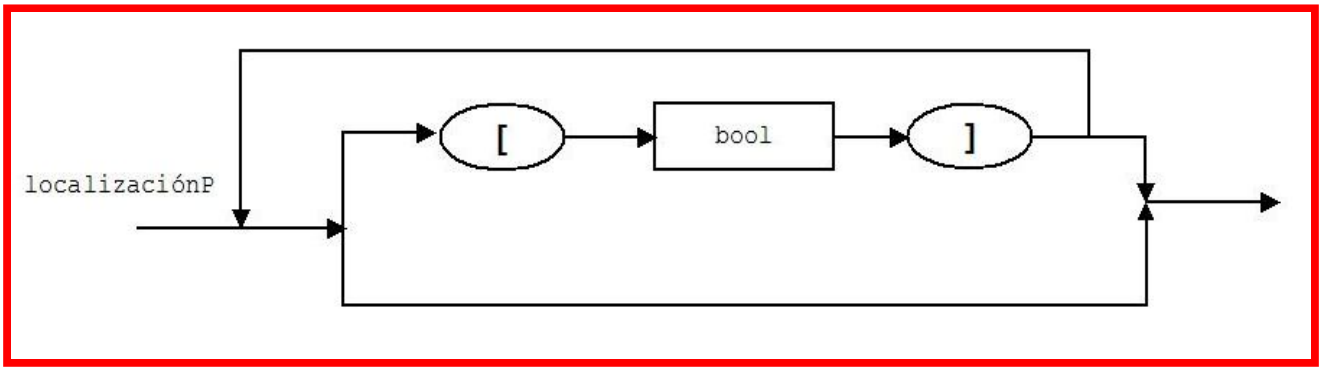
40) lista_paramP → **,** bool lista_paramP / ε



41) localización → **[** bool **]** localizacionP



42) localizacionP → **[** bool **]** localizacionP | ε



Definición Dirigida por Sintaxis.

En la fase del compilador de Análisis Semántico se realiza una Tabla de Definición Dirigida por Sintaxis, en la cual empezamos a añadir más información al árbol sintáctico. Cada símbolo no terminal puede tener asociados uno o más atributos, estos pueden ser de cualquier tipo de dato, el significado que se le atribuya depende de quien lo programe.

Existen dos tipos de atributos:

- **Atributos Heredados:** Son aquellos de los cuales su valor depende de los atributos asociados con el símbolo padre o los símbolos hermanos.

- **Atributos Sintetizados:** Son aquellos atributos asociados a los encabezados de las producciones y su valor es calculado a partir de los nodos hijos.

El siguiente cuadro representa la gramática con atributos, a cada regla de producción se le puede asociar cero o más reglas semánticas.

Reglas de producción	Reglas semánticas
programa → declaraciones funciones	PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0

declaraciones → tipo lista var **;** declaraciones	lista_var.tipo = tipo.tipo
declaraciones → ε	//vacío
tipo → basico compuesto	compuesto.base = basico.tipo tipo.tipo = compuesto.tipo
basico →**int**	basico.tipo=int
basico →**float**	basico.tipo=float
basico →**char**	basico.tipo=char
basico →**double**	basico.tipo=double
basico →**void**	basico.tipo=void
compuesto → **[numero]** compuesto1	compuesto.tipo = PilaTT.top().insertar(”array”, num.val, compuesto1.tipo) compuesto1.base = compuesto.base
compuesto → ε	compuesto.tipo = compuesto.base
lista var → **id** lista_varP	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, lista var.tipo, dir, ”var”, NULO) dir = dir + PilaTT.top().getTam(tipo.tipo) Sino error(”El id ya está declarado”) ´ FinSi lista_varP.tipo = lista_var.tipo
lista_varP → **, id** lista_varP1	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, lista_varP.tipo, dir, ”var”, NULO) dir = dir + PilaTT.top().getTam(lista_varP.tipo) Sino error(”El id ya esta declarado”) FinSi lista_varP1.tipo = lista_varP.tipo
lista_varP → ε	//vacío
funciones →**func** tipo **id (** argumentos **)** bloque funciones	ListaRetorno = NULO PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos) PilaDir.push(dir) dir = 0 Si ! PilaTS.top().buscar(id) Entonces Si equivalentesLista(ListaRetorno, tipo.tipo) Entonces PilaTS.top().insertar(id, tipo.tipo, –, ’func’,argumentos.lista) genCod(label(id)) bloque.sig = nuevaEtq() genCod(label(bloque.sig)) Sino error(”Los tipos de retorno no coinciden con el tipo de la función”) FinSi Sino error(”El id ya está declarado”)

	FinSi PilaTS.pop() PilaTT.pop() dir = PilaDir.pop()
funciones → ε	//vacío
argumentos → lista args	argumentos.lista = lista_args
argumentos → ε	argumentos.lista = NULO
lista args → tipo **id** lista_argsP	lista_argsP.listaHer = nuevaListaArgs() lista_args.lista = lista_argsP.listaSin lista_argsP.listaHer.agregar (tipo.tipo) Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, tipo.tipo, dir, ”arg”, NULO) dir = dir + PilaTT.top().getTam(tipo.tipo) Sino error(”El id ya esta declarado”) ´ FinSi
lista_argsP → **,** tipo **id** lista_argsP1	lista_argsP1.listaHer = lista_argsP.listaHer lista_argsP.listaSin = lista_argsP1.listaSin lista_argsP1.listaHer.agregar (tipo.tipo) Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, tipo.tipo, dir, ”arg”, NULO) dir = dir + PilaTT.top().getTam(tipo) Sino error(”El id ya esta declarado”) ´ FinSi
lista_argsP → ε	lista_argsP.listaSin = lista_argsP.listaHer
bloque → *{* declaraciones instrucciones *}*	
instrucciones → sentencia instruccionesP	sentencia.sig = nuevaEtq() genCod(label(sentencia.sig))
instruccionesP → sentencia instruccionesP1	sentencia.sig = nuevaEtq() genCod(label(sentencia.sig))
instruccionesP → ε	instruccionesP.sig = NULO
sentencia → parte_izquierda **=** bool**;**	Si equivalentes(parte_izquierda.tipo, bool.tipo) Entonces d1 = reducir(bool.dir, bool.tipo,parte_izquierda.tipo)

	genCod(parte_izquierda.dir '=' d1) Sino error("Tipos incompatibles") FinSi
sentencia → **if(** bool **)** sentencia1 Sent	bool.vddr = nuevaEtiqueta() bool.fls = nuevoIndice() sentencia1.sig = sentencia.sig Sent.sig = sentencia.sig Sent.lista_indices = nuevaListaIndices() Sent.lista_indices.agregar(bool.fls) genCod(label(bool.vddr))
Sent → **else** sentencia	sentencia.sig = Sent.sig genCod('goto' sentencia.sig) genCod(label(Sent.lista_indices[0])) reemplazarIndices(Sent.lista_indices, nuevaEtiqueta(), cuadruplas)
Sent → ε	reemplazarIndices(Sent.lista_indices, Sent.sig, cuadruplas)
sentencia → **while(** bool **)** sentencia1	sentencia1 .sig = nuevaEtiqueta() bool.vddr = nuevaEtq() bool.fls = sentencia.sig genCod(label(sentencia1 .sig)) genCod(label(bool.vddr)) genCod('goto' sentencia1 .sig)
sentencia → **do** sentencia1 **while(** bool **)**	bool.vddr = nuevaEtiquetal() bool.fls = sentencia.sig sentencia1 .sig = nuevaTemporal() genCod(label(bool.vrdd)) genCod(label(sentencia1.sig))
sentencia → **break ;**	genCod(goto sentencia.sig)
sentencia → **return** Func_ret	ListaRetorno.agregar(Func_ret.tipo)
sentencia → bloque	bloque.sig = sentencia.sig
sentencia → **switch(** bool **)** *{* casos *}*	

	casos.etqprueba = nuevaEtq() genCode('goto' casos.etqprueba) casos.sig = sentencia.sig casos.id = bool.dir genCode(label(casos.etqprueba)) genCode(casos.prueba)
sentencia → **print** exp **;**	genCode ('print' exp.dir)
sentencia → **scan** parte_izquierda	genCode ('scan' parte_izquierda.dir)
Func_ret → **;**	Func_ret.tipo = void genCod('return')
Func_ret → exp;	Func_ret.tipo = exp.tipo genCod('return' exp.dir)
casos → caso casos1	$casos_1$.sig = casos.sig caso.sig = casos.sig casos.prueba = caso.prueba // $casos_1$.prueba
casos → ε	casos.prueba = ''
casos → predeterminado	casos.prueba = predeterminado.prueba predeterminado.sig =casos.sig
caso → **case numero:** instrucciones	caso.inicio = nuevaEtq() caso.prueba = genCod(if caso.id '==' numero.lexval 'goto' caso.inicio) genCode(label(caso.inicio))
predeterminado → **default :** instrucciones	predeterminado.inicio = nuevaEtq() instrucciones.sig = predeterminado.sig predeterminado.prueba = genCod('goto' predeterminado.inicio) genCode(label(predeterminado.inicio))
parte_izquierda → **id** parte_izquierdaP	Si PilaTS.top().buscar(id) Entonces parte_izquierdaP.base = id.lexval parte_izquierda.dir = parte_izquierdaP.dir parte_izquierda.tipo = parte_izquierdaP.tipo Sino error("El id no está declarado"); FinSi
parte_izquierdaP → localización	localizacion.base = parte_izquierdaP.base parte_izquierdaP.dir = localizacion.dir

	parte_izquierdaP.tipo = localizacion.tipo

parte_izquierdaP → ε	Si PilaTS.top().buscar(parte_izquierdaP.base) Entonces
	  parte_izquierdaP.dir = parte_izquierdaP.base
	  parte_izquierdaP.tipo =PilaTS.top().getTipo(parte_izquierdaP.dir)
	Sino
	  error(”El id no está declarado”)
	FinSi

bool → comb boolP	comb.vddr = bool.vddr comb.fls = nuevoIndice(); boolP.tipoHer = comb.tipo boolP.vddr = bool.vddr boolP.fls = bool.fls boolP.listaIndices = nuevaListaIndices() boolP.listaIndices.agregar(comb.fls) genCode(label(comb.fls) bool.tipo=boolP.tipoSin bool.dir=comb.dir

boolP → **||** comb boolP1	comb.vddr = boolP1.vddr
	boolP.tipoSin = boolP1.tipoSin
	boolP.dirSin = comb.dir
	comb.fls = nuevoIndice();
	
	Si equivalentes(boolP.tipoHer, comb.tipo) Entonces
	  boolP1 .tipoHer = comb.tipo
	  boolP1 .vddr = boolP.vddr
	  boolP1 .fls = boolP.fls
	  boolP1.lista_indices = boolP.lista_indices
	  boolP1.lista_indices.agregar(comb.fls)
	  genCod(label(comb.fls))
	Sino
	  error("Los tipos no son compatibles")
	FinSi

boolP → ε	reemplazarIndices(boolP.lista_indices, boolP.fls, cuadruplas) boolP.tipoS = int boolP.dirS = boolP.dirH
comb → igualdad combP	igualdad.vddr = nuevoIndice() igualdad.fls =comb.fls combP.tipoH = igualdad.tipo combP.vddr = comb.vddr combP.fls = comb.fls combP.lista_indices = nuevaListaIndices() combP.lista_indices.agregar(igualdad .vddr) comb.tipo = combP.tipoS comb.dir =igualdad.dir genCod(label(igualdad.vddr))
combP → **&&** igualdad combP1	Si equivalentes(combP.tipoH, igualdad.tipo) Entonces

	igualdad.vddr = nuevoIndice() igualdad.fls = combP.fls combP1 .tipoH = igualdad.tipo combP1 .vddr = combP.vddr combP1 .fls = combP.fls combP1.lista_indices = combP1.lista_indices combP1.lista_indices.agregar(igualdad.vddr) combP.tipoS = combP1.tipoS genCod(label(igualdad.vddr)) Sino error(”Tipos incompatibles”) FinSi
combP → ε	reemplazarIndices(combP.lista_indices, combP.vddr, cuadruplas) combP.tipoS = int combP.dirS = combP.dirH
igualdad → rel igualdadP	rel.vddr = igualdad.vddr rel.fls = igualdad.fls igualdadP.tipoHer = rel.tipo igualdadP.dirHer = rel.dir igualdad.tipo=igualdadP.tipoSin igualdad.dir=igualdadP.dirSin
igualdadP → **==** rel igualdadP1	rel.vddr = igualdadP.vddr rel.fls = igualdadP.fls Si equivalentes(igualdadP.tipoHer, rel.tipo) Entonces igualdadP1.tipoHer =maximo(igualdadP.tipoHer, rel.tipo) igualdadP1.dirHer = nuevaTemporal() d1 = ampliar(igualdadP.dirHer, igualdadP.tipoHer, igualdadP1.tipoHer) d2 = ampliar(rel.dir, rel.tipo, igualdadP1.Her) igualdadP.tipoSin = igualdadP1.tipoSin igualdadP.dirSin = igualdadP1.dirSin genCod (igualdadP1.dir '=' d1 '==' d2) Sino error(”Tipos incompatibles”) FinSi
igualdadP → **!=** rel igualdadP1	rel.vddr = igualdadP.vddr rel.fls = igualdadP.fls Si equivalentes(igualdadP.tipoHer, rel.tipo) Entonces igualdadP1.tipoHer =maximo(igualdadP.tipoHer, rel.tipo) igualdadP1.dirHer = nuevaTemporal() d1 = ampliar(igualdadP.dirHer, igualdadP.tipoHer, igualdadP1.tipoHer) d2 = ampliar(rel.dir, rel.tipo, igualdadP1.Her) igualdadP.tipoSin = igualdadP1.tipoSin igualdadP.dirSin = igualdadP1.dirSin

	genCod (igualdadP1.dir '=' d1 '!=' d2) Sino error(”Tipos incompatibles”) FinSi
igualdadP → ε	igualdadP.tipoSin = igualdadP.tipoHer igualdadP.dirSin = igualdadP.tipoHer
rel →exp relP	relP.tipoHer = exp.tipoSin relP.dirHer = exp.dirSin rel.tipoSin = relP.tipoSin rel.dirSin = relP.dirSin relP.vddr = rel.vddr relP.fls = rel.fls
relP → **<=** exp	exp.vddr = relP.vddr exp.fls = relP.fls Sí equivalentes(relP.tipoHer, exp.tipo) Entonces tipoTemp =maximo(relP.tipoHer, exp.tipo) relP.dirSin = nuevaTemporal() relP.tipo = int d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp) d2 = ampliar(exp.dir, exp.tipo, tipoTemp) genCod (relP.dir '=' d1.dir '<=' d2.dir) genCod (‘if’ relP.dir ‘goto’ relP.vddr) genCod (‘goto’ relP.fls) Sino error(”Tipos incompatibles”) FinSi
relP → **>=** exp	exp.vddr = relP.vddr exp.fls = relP.fls Sí equivalentes(relP.tipoHer, exp.tipo) Entonces tipoTemp =maximo(relP.tipoHer, exp.tipo) relP.dirHer = nuevaTemporal() relP.tipo = int d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp) d2 = ampliar(exp.dir, exp.tipo, tipoTemp) genCod (relP.dir '=' d1.dir '>=' d2.dir) genCod (‘if’ relP.dir ‘goto’ relP.vddr) genCod (‘goto’ relP.fls) Sino error(”Tipos incompatibles”) FinSi
relP → **<** exp	exp.vddr = relP.vddr exp.fls = relP.fls Sí equivalentes(relP.tipoHer, exp.tipo) Entonces tipoTemp =maximo(relP.tipoHer, exp.tipo)

	relP.dirHer = nuevaTemporal() relP.tipo = int d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp) d2 = ampliar(exp.dir, exp.tipo, tipoTemp) genCod (relP.dir '=' d1.dir '<' d2.dir) genCod ('if' relP.dir 'goto' relP.vddr) genCod ('goto' relP.fls) Sino error("Tipos incompatibles") FinSi
relP → **>** exp	exp.vddr = relP.vddr exp.fls = relP.fls Sí equivalentes(relP.tipoHer, exp.tipo) Entonces tipoTemp =maximo(relP.tipoHer, exp.tipo) relP.dirHer = nuevaTemporal() relP.tipo = int d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp) d2 = ampliar(exp.dir, exp.tipo, tipoTemp) genCod (relP.dir '=' d1.dir '>' d2.dir) genCod ('if' relP.dir 'goto' relP.vddr) genCod ('goto' relP.fls) Sino error("Tipos incompatibles") FinSi
relP → ε	relP.tipoSin=relP.tipoHer relP.dirSin=relP.dirHer
exp → term expP	 expP.tipoHer = term.tipoSin expP.dirHer = term.dirSin exp.tipoSin = expP.tipoSin exp.dirSin = expP.dirSin
expP → +term expP1	Si equivalentes(expP.tipoHer, term.tipo) Entonces expP1.tipoHer =maximo(expP.tipoHer, term.tipo) expP1.dirHer = nuevaTemporal() d1 = ampliar(expP.dirHer, expP.tipoHer, expP1.tipoHer) d2 = ampliar(term.dir, term.tipo, expP1.Her) expP.tipoSin = expP1.tipoSin expP.dirSin = expP1.dirSin genCod (expP1.dirHer '=' d1 '+' d2) Sino error("Los tipos no son compatibles") FinSi
expP → -term expP1	Si equivalentes(expP.tipoHer, term.tipo) Entonces expP1.tipoHer =maximo(expP.tipoHer, term.tipo) expP1.dirHer = nuevaTemporal() d1 = ampliar(expP.dirHer, expP.tipoHer, expP1.tipoHer)

	d2 = ampliar(term.dir, term.tipo, expP1.Her) expP.tipoSin = expP1.tipoSin expP.dirSin = expP1.dirSin genCod (expP1.dirHer '=' d1 '-' d2) Sino error("Los tipos no son compatibles") FinSi
expP → ε	expP.tipoSin = expP.tipoHer expP.dirSin = expP.dirHer
term → unario termP	termP.tipoHer = unario.tipoSin termP.dirHer = unario.dirSin term.tipoSin = termP.tipoSin term.dirSin = termP.dirSin
termP → * unario termP1	Si equivalentes(termP.tipoHer, unario.tipo) Entonces termP1.tipoHer =maximo(termP.tipoHer, unario.tipo) termP1.dirHer = nuevaTemporal() d1 = ampliar(termP.dirHer, termP.tipoHer, termP1.tipoHer) d2 = ampliar(unario.dir, unario.tipo, termP1.Her) termP.tipoSin = termP1.tipoSin termP.dirSin = termP1.dirSin genCod (termP1.dir '=' d1 '*' d2) Sino error("Los tipos no son compatibles") FinSi
termP → / unario termP 1	Si equivalentes(termP.tipoHer, unario.tipo) Entonces termP1.tipoHer =maximo(termP.tipoHer, unario.tipo) termP1.dirHer = nuevaTemporal() d1 = ampliar(termP.dirHer, termP.tipoHer, termP1.tipoHer) d2 = ampliar(unario.dir, unario.tipo, termP1.Her) termP.tipoSin = termP1.tipoSin termP.dirSin = termP1.dirSin genCod (termP1.dir '=' d1 '/' d2) Sino error("Los tipos no son compatibles") FinSi
termP → % unario termP1	Si termP.tipoHer == int Y unario.tipoSin == int Entonces termP1.tipoHer = int termP1.dirHer = nuevaTemporal() termP.tipoSin = int genCod (termP1.dirHer '=' termP.dirHer % unario.dir) Sino error("Los tipos no son compatibles") FinSi
termP → ε	termP.tipoSin = termP.tipoHer termP.dirSin = termP.dirHer
unario → !unario1	unario.dir = nuevaTemporal() unario.tipo = unario1.tipo genCod(unario.dir '=' '!' unario1.dir)
unario → – unario1	unario.dir = nuevaTemporal() unario.tipo = unario1.tipo genCod(unario.dir '=' '!' unario1.dir)

unario → factor	unario.dir = factor.dir unario.tipo = factor.tipo
factor → **id** factorP	factorP.base = id.lexval factor.dir = factorP.dir factor.tipo = factorP.tipo
factor → **(**bool**)**	factor.tipo = bool.tipo factor.dir = bool.dir
factor → **numero**	factor.dir = numero.lexval factor.tipo = numero.tipo (lex)
factor → **cadena**	TablaCadenas.agregar(cadena.lexval) factor.dir =TablaCadenas.getUltimaPos() factor.tipo = cadena
factor → **true**	factor.dir = 'true' factor.tipo = int
factor → **false**	factor.dir = 'false' factor.tipo = int
factorP→localización	Localizacion.base = factorP.base factorP.dir = nuevaTemporal() factorP.tipo = localización.tipo genCod(factorP.dir '=' factorP.base '[' localizacion.dir']')
factorP → ε	factorP.dir =factorP.base Si PilaTS.top().buscar(factorP.base) Entonces factorP.tipo=PilaTS.top().getTipo(factorP.dir) Sino factorP.tipo=PilaTS.fondo().getTipo(factorP.dir) FinSi
factorP→ **(**parametros**)**	Si PilaTS.fondo().buscar(factorP.base) Entonces Si PilaTS.fondo().getVar(factorP.base) = 'func' Entonces Si equivalenteListas(PilaTS.fondo().getArgs(factorP.base), parametros.lista) Entonces factorP.tipo = PilaTS.top().getTipo(id) factorP.dir = nuevaTemporal() genCod(factorP.dir '=' 'call' id ',' parametros.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
parametros → lista param	parametros.lista = lista_param
parametros → ε	parametros.lista = NULO

lista param → bool lista_paramP	lista_paramP.listaHer = nuevaListaArgs() lista_param.lista = lista_paramP.listaSin lista_param.listaHer.agregar(bool.tipo) genCode('param' bool.dir)
lista_paramP → , bool lista_paramP1	lista_paramP1.listaHer = lista_paramP.listaHer lista_paramP.listaSin = lista_paramP1.listaSin lista_paramP1.listaHer.agregar (bool.tipo) genCode('param' bool.dir)
lista_paramP → ε	lista_paramP.listaSin = lista_paramP.listaHer
localización →[bool] localizacionP	Si PilaTS.top().buscar(localizacion.base) Entonces Si bool.tipo = int Entonces tipoTmp = PilaTS.top().getTipo(localizacion.base) Si PilaTT.top().getNombre(tipoTmp)='array' Entonces localizacionP.tipo = PilaTT.top().getTipoBase(tipoTmp) localizacionP.dir = nuevaTemporal() localizacionP.tam = PilaTT.top().getTam(localizacionP.tipo) genCod(localizacion.dir '=' bool.dir '*' localizacion.tam) localizacion.dir = localizacionP.dirS localizacion.tipo = localizacionP.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
localizacionP →[bool] localizacionP1	Si bool.tipo = int Entonces Si PilaTT.top.getNombre(localizacionP.tipo)="array" Entonces localizacionP1.tipo = PilaTT.top().getTipoBase(localizacionP.tipo) dirTmp = nuevaTemporal() localizacionP1.dir = nuevaTemporal() localizacionP1.tam = PilaTT.top().getTam(localizacionP.tipo) genCod(dirTmp='bool.dir'*' localizacionP1.tam genCod(localizacionP1.dir '='localizacionP.dir'+'dirTmp) localizacionP.dirSin = localizacionP1.dirSin localizacionP.tipoSin = localizacionP1.tipoSin Sino error("El id no es un arreglo") FinSi Sino error("El índice del arreglo debe ser entero")

	FinSi
localizacionP → ε	localizacionP.dirS=localizacionP.dir localizacionP.tipoS=localizacion.tipo

Esquema de Traducción.

Esquema de traducción
programa → { PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0 } declaraciones funciones
declaraciones → tipo { lista_var.tipo = tipo.tipo } lista_var ; declaraciones declaraciones → ε { getCod(label(declaraciones.sig)) }
tipo →basico { compuesto.base = basico.tipo tipo.tipo = compuesto.tipo } compuesto
basico→ int{ basico.tipo=int } | float { basico.tipo=float } | char { basico.tipo=char } double { basico.tipo=double } | void { basico.tipo=void }
compuesto → **[numero]** compuesto1 { compuesto.tipo = PilaTT.top().insertar(”array”, num.val, compuesto1.tipo) compuesto1.base = compuesto.base } | { compuesto.tipo = compuesto.base }
lista var → id { lista_varP.tipo = lista_var.tipo Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insertar(id, lista var.tipo, dir, ”var”, NULO) dir = dir + PilaTT.top().getTam(tipo.tipo) Sino error(”El id ya está declarado”) ´ FinSi } lista_varP
lista_varP → **, id** {

```
 lista_varP1.tipo = lista_varP.tipo
Si ! PilaTS.top().buscar(id) Entonces
    PilaTS.top().insertar(id, lista_varP.tipo, dir, ”var”, NULO)
    dir = dir + PilaTT.top().getTam(lista_varP.tipo)
Sino
     error(”El id ya esta declarado”)
FinSi } lista_varP1
```

lista_varP → ε { }

funciones → **func** tipo **id** {

```
ListaRetorno = NULO
PilaTS.push(nuevaTablaSimbolos)
PilaTT.push(nuevaTablaTipos)
PilaDir.push(dir)
dir = 0
genCod(label(id))

} ( argumentos )  bloque {
PilaTS.pop() PilaTT.pop()
dir = PilaDir.pop()

 Si ! PilaTS.top().buscar(id) Entonces
   Si equivalentesLista(ListaRetorno, tipo.tipo) Entonces
      bloque.sig = nuevaEtq()
      genCod(label(bloque.sig))
   Sino
     error(”Los tipos de retorno no coinciden con el tipo de la función”)
   FinSi
Sino
   error(”El id ya está declarado”)
FinSi }  funciones
```

funciones → ε { }

```
argumentos→  { argumentos.lista = lista_args }lista args
                    |   { argumentos.lista = NULO }
```

```
lista args→ tipo id {
                        lista_argsP.listaHer = nuevaListaArgs()
                        lista_argsP.listaHer.agregar (tipo.tipo)
                       Si ! PilaTS.top().buscar(id) Entonces
                            PilaTS.top().insertar(id, tipo.tipo, dir, ”arg”, NULO)
                            dir = dir + PilaTT.top().getTam(tipo.tipo)
                       Sino
                            error(”El id ya esta declarado”) ´
                       FinSi   } lista_argsP {  lista_args.lista = lista_argsP.listaSin   }
```

lista_argsP → , tipo id {
lista_argsP1.listaHer = lista_argsP.listaHer
lista_argsP1.listaHer.agregar (tipo.tipo)

Si ! PilaTS.top().buscar(id) Entonces
PilaTS.top().insertar(id, tipo.tipo, dir, ”arg”, NULO)
dir = dir + PilaTT.top().getTam(tipo.tipo)
Sino
error(”El id ya esta declarado”) ´
FinSi } lista_argsP1 { lista_argsP.listaSin = lista_argsP1.listaSin }

| { lista_argsP.listaSin = lista_argsP.listaHer }

---

bloque *{* declaraciones instrucciones *}*

---

instrucciones → { sentencia.sig = nuevaEtq() } sentencia { genCod(label(sentencia.sig)) } instruccionesP

---

instruccionesP → { sentencia.sig = nuevaEtq() } sentencia { genCod(label(sentencia.sig)) } instruccionesP1
| { instruccionesP.sig = NULO }

---

sentencia → parte_izquierda **=** bool**;** {
Si equivalentes(parte_izquierda.tipo, bool.tipo) Entonces
d1 = reducir(bool.dir, bool.tipo,parte_izquierda.tipo)
genCod(parte_izquierda.dir ’=’ d1)
Sino
error(”Tipos incompatibles”)
FinSi
}

---

sentencia → **if(** bool {

bool.vddr = nuevaEtiqueta()

bool.fls = nuevoIndice() } **)**

{ genCod(label(bool.vddr))

sentencia1.sig = sentencia.sig } sentencia1 { Sent.sig = sentencia.sig

Sent.lista_indices.agregar(bool.fls) } Sent

Sent → **else** {sentencia.sig = Sent.sig
genCod(‘goto’ sentencia.sig)
genCod(label(Sent.lista_indices[0])) } sentencia {
reemplazarIndices(Sent.lista_indices, nuevaEtiqueta(), cuadruplas) }

| { reemplazarIndices(Sent.lista_indices, Sent.sig, cuadruplas) }

sentencia → **while(** bool { sentencia1 .sig = nuevaEtiqueta()
bool.vddr = nuevaEtq()
bool.fls = sentencia.sig
genCod(label(sentencia1 .sig))
} **)** sentencia1 { genCod(label(bool.vddr)
genCod('goto' sentencia1 .sig) }

sentencia →**do** { bool.vddr = nuevaEtiquetal()
bool.fls = sentencia.sig
sentencia1.sig = nuevaEtiqueta()
genCod(label(bool.vrdd)) } sentencia1 { genCod(label(sentencia1.sig)) } **while(** bool **)**

sentencia → **break;** { genCod(goto sentencia.sig) }

| **return** Func_ret {ListaRetorno.agregar(Func_ret.tipo) }

| { bloque.sig = sentencia.sig } bloque

sentencia → **switch(** bool { casos.etqprueba = nuevaEtq()
genCode('goto' casos.etqprueba)
casos.sig = sentencia.sig }
) *{* casos { casos.id = bool.dir
genCode(label(casos.etqprueba))
genCode(casos.prueba) } *}*

sentencia→ **print** exp { genCode ('print' exp.dir) } ;

| **scan** parte_izquierda { genCode ('scan' parte_izquierda.dir) }

Func_ret →{ Func_ret.tipo = void
genCod('return') } **;**
| **exp;** { Func_ret.tipo = exp.tipo
genCod('return' exp.dir) }

casos → caso {
casos1.sig = casos.sig
caso.sig = casos.sig
casos.prueba = caso.prueba
} casos1 { // casos1.prueba }

| predeterminado { casos.prueba = predeterminado.prueba
predeterminado.sig =casos.sig }

casos → ε
caso → **case numero:** { caso.inicio = nuevaEtq()
instrucciones.sig = caso.sig
caso.prueba = genCod(if caso.id '==' numero.lexval 'goto' caso.inicio)
genCode(label(caso.inicio)) } instrucciones

predeterminado→ **default :** { predeterminado.inicio = nuevaEtq()
instrucciones.sig = predeterminado.sig
predeterminado.prueba = genCod('goto' predeterminado.inicio)
genCode(label(predeterminado.inicio)) } instrucciones

parte_izquierda →**id** {
parte_izquierdaP.base = id.lexval
} parte_izquierdaP {
parte_izquierda.dir = parte_izquierdaP.dir
parte_izquierda.tipo = parte_izquierdaP.tipo
}

parte_izquierdaP → { localizacion.base = parte_izquierdaP.base }
localización {
parte_izquierdaP.dir = localizacion.dir
parte_izquierdaP.tipo = localizacion.tipo }

| { Si PilaTS.top().buscar(parte_izquierdaP.base) Entonces
parte_izquierdaP.dir = parte_izquierdaP.base
parte_izquierdaP.tipo =PilaTS.top().getTipo(parte_izquierdaP.dir)
Sino
error("El id no está declarado")
FinSi }

bool→{ comb.vddr = bool.vddr
comb.fls =nuevoIndice()
} comb { boolP.tipoH = comb.tipo
bool.dir=comb.dir
boolP.lista_indices = nuevaListaIndices()
boolP.lista_indices.agregar(comb.fls)
genCod(label(comb.fls)) } boolP { bool.tipo = boolP.tipoS }

boolP→|| { comb.vddr = bool.vddr
boolP.dirSin = comb.dir
comb.fls =nuevoIndice() } comb
{ Si equivalentes(boolPtipoH, comb.tipo) Entonces
boolP1 .tipoHer = comb.tipo
boolP1 .vddr = bool.vddr
boolP1 .fls = bool.fls
boolP1.lista_indices = boolP1.lista_indices

```
                boolP1.lista_indices.agregar(comb.fls)
                genCod(label(comb .fls))
        Sino
                 error(”Tipos incompatibles”)
        FinSi
        }  boolP1 { boolP.tipoS = boolP1.tipoS }

        |  { reemplazarIndices(boolP.lista_indices, boolP.fls, cuadruplas)
           boolP.tipoS = int }
```

```
comb → {   igualdad.vddr = nuevoIndice()
           igualdad.fls =comb.fls   }
          igualdad {  combP.lista_indices = nuevaListaIndices()
                 combP.lista_indices.agregar(igualdad .vddr)
                      genCod(label(igualdad.vddr))
                comb.dir =igualdad.dir }
             combP { comb.tipo = combP.tipoS

                          combP.tipoH = igualdad.tipo }
```

```
combP →  && igualdad {
Si equivalentes(combP.tipoH, igualdad.tipo) Entonces
    igualdad.vddr = nuevoIndice()
    igualdad.fls = comb.fls
    combP1 .tipoH = igualdad.tipo
    combP1.vddr = combP.vddr
    combP1.fls = combP.fls
    combP1.lista_indices =  combP1.lista_indices
    combP1.lista_indices.agregar(igualdad.vddr)
    genCod(label(igualdad.vddr))
Sino
    error(”Tipos incompatibles”)
FinSi  }  combP1  { combP.tipoS = combP1.tipoS }

|  { reemplazarIndices(combP.lista_indices, combP.vddr, cuadruplas)
     combP.tipoS = int }
```

```
igualdad →{   rel.vddr = igualdad.vddr
              rel.fls = igualdad.fls }
         rel {   igualdadP.tipoHer = rel.tipo
              igualdadP.dirHer = rel.dir }
          igualdadP1 {  igualdad.tipo=igualdadP.tipoSin
                        igualdad.dir=igualdadP.dirSin }

  |  { igualdadP.tipoSin = igualdadP.tipoHer }
```

```
rel →exp { relP.tipoHer = exp.tipoSin
         relP.dirHer = exp.dirSin
```

```
        } relP {  rel.tipoSin = relP.tipoSin
                  rel.dirSin = relP.dirSin }
```

```
relP → <=  exp  {
Sí equivalentes(relP.tipoHer, exp.tipoSin) Entonces
    tipoTemp =maximo(relP.tipoHer, exp.tipoSin)
    relP.dirSin = nuevaTemporal()
    relP.tipo = int
     d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp)
     d2 = ampliar(exp.dirSin, exp.tipoSin, tipoTemp)
     genCod ( relP.dir '=' d1.dir '<=' d2.dir)
     genCod (‘if’ relP.dir ‘goto’ relP.vddr)
     genCod (‘goto’ relP.fls)
Sino
        error(”Tipos incompatibles”)
FinS }
```

```
relP → >=  exp {
Sí equivalentes(relP.tipoHer, exp.tipoSin) Entonces
    tipoTemp =maximo(relP.tipoHer, exp.tipoSin)
    relP.dirHer = nuevaTemporal()
     relP.tipo = int
     d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp)
     d2 = ampliar(exp.dirSin, exp.tipoSin, tipoTemp)
     genCod ( relP.dir '=' d1.dir '>=' d2.dir)
     genCod (‘if’ relP.dir ‘goto’ relP.vddr)
     genCod (‘goto’ relP.fls)
Sino
        error(”Tipos incompatibles”)
FinSi  }
```

```
relP → <  exp {
Sí equivalentes(relP.tipoHer, exp.tipoSin) Entonces
    tipoTemp =maximo(relP.tipoHer, exp.tipoSin)
    relP.dirHer = nuevaTemporal()
     relP.tipo = int
     d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp)
     d2 = ampliar(exp.dirSin, exp.tipoSin, tipoTemp)
     genCod ( relP.dir '=' d1.dir '<' d2.dir)
     genCod (‘if’ relP.dir ‘goto’ relP.vddr)
     genCod (‘goto’ relP.fls)
Sino
        error(”Tipos incompatibles”)
FinSi }
```

```
relP → > {
Sí equivalentes(relP.tipoHer, exp.tipoSin) Entonces
    tipoTemp =maximo(relP.tipoHer, exp.tipoSin)
    relP.dirHer = nuevaTemporal()
     relP.tipo = int
     d1 = ampliar(relP.dirHer, relP.tipoHer, tipoTemp)
     d2 = ampliar(exp.dirSin, exp.tipoSin, tipoTemp)
     genCod ( relP.dir '=' d1.dir '>' d2.dir)
     genCod (‘if’ relP.dir ‘goto’ relP.vddr)
     genCod (‘goto’ relP.fls)
Sino
        error(”Tipos incompatibles”)
FinSi } exp

| { relP.tipoSin=relP.tipoHer
  relP.dirSin=relP.dirHer }
```

```
exp →expP { expP.tipoHer = term.tipoSin
                expP.dirHer = term.dirSin }
        term { exp.tipoSin = expP.tipoSin
                exp.dirSin = expP.dirSin }
```

```
expP → +term {
                Si equivalentes(expP.tipoHer, term.tipoSin) Entonces
                      expP1.tipoHer =maximo(expP.tipoHer, term.tipoSin)
                      expP1.dirHer = nuevaTemporal()
                      d1 = ampliar(expP.dirHer, expP.tipoHer, expP1.tipoHer)
                      d2 = ampliar(term.dirHer, term.tipoHer, expP1.Her)
                Sino error("Los tipos no son compatibles")
                 FinSi
} expP1 {   expP.tipoSin = expP1.tipoSin
                expP.dirSin = expP1.dirSin }
```

```
expP → - term {
                Si equivalentes(expP.tipoHer, term.tipoSin) Entonces
                    expP1.tipoHer =maximo(expP.tipoHer, term.tipoSin)
                    expP1.dirHer = nuevaTemporal()
                    d1 = ampliar(expP.dirHer, expP.tipoHer, expP1.tipoHer)
                    d2 = ampliar(term.dirHer, term.tipoHer, expP1.Her)
                    genCod ( expP1.dir '=' d1 '-' d2)
                Sino
                     error("Los tipos no son compatibles")
                FinSi } expP1 {   expP.tipoSin = expP1.tipoSin
                                  expP.dirSin = expP1.dirSin }
```

```
        | { expP.tipoSin = expP.tipoHer
            expP.dirSin = expP.dirHer }
```

```
term → unario { termP.tipoHer = unario.tipoSin
                termP.dirHer = unario.dirSin }
       termP { term.tipoSin = termP.tipoSin
               term.dirSin = termP.dirSin }
```

```
termP → * unario {

Si equivalentes(termP.tipoHer, unario.tipoSin) Entonces
    termP1.tipoHer =maximo(termP.tipoHer, unario.tipoSin)
    termP1.dirHer = nuevaTemporal()
    d1 = ampliar(termP.dirHer, termP.tipoHer, termP1.tipoHer)
    d2 = ampliar(unario.dirHer, unario.tipoHer, termP1.Her)
     genCod ( termP1.dir '=' d1 '*' d2)
Sino
    error("Los tipos no son compatibles")
FinSi } termP1 { termP.tipoSin = termP1.tipoSin
         termP.dirSin = termP1.dirSin }
```

```
termP → / unario {

Si equivalentes(termP.tipoHer, unario.tipoSin) Entonces
    termP1.tipoHer =maximo(termP.tipoHer, unario.tipoSin)
    termP1.dirHer = nuevaTemporal()
    d1 = ampliar(termP.dirHer, termP.tipoHer, termP1.tipoHer)
    d2 = ampliar(unario.dirHer, unario.tipoHer, termP1.Her)
    genCod ( termP1.dir '=' d1 '/' d2)
Sino error("Los tipos no son compatibles")
FinSi }  termP1 { termP.tipoSin = termP1.tipoSin
                  termP.dirSin = termP1.dirSin }
```

```
termP → % unario {

Si termP.tipoHer == int Y unario.tipoSin == int Entonces
    termP1.tipoHer = int
    termP1.dirHer = nuevaTemporal()
    termP.tipoSin = int
   genCod ( termP1.dirHer '=' termP.dirHer % unario.dir)
Sino error("Los tipos no son compatibles")
FinSi }   termP1

| { termP.tipoSin = termP.tipoHer
  termP.dirSin = termP.dirHer }
```

```
unario → !unario1 { unario.dir = nuevaTemporal()
                    unario.tipo = unario1.tipo
                    genCod(unario.dir '=' '!' unario1.dir) }
```

```
unario → - unario1 {  unario.dir = nuevaTemporal()
```

unario.tipo = unario1.tipo genCod(unario.dir '=' '!' unario1.dir) }
unario → factor { unario.dir = factor.dir unario.tipo = factor.tipo }
factor → **id** { factorP.base = id.lexval } factorP { factor.dir = factorP.dir factor.tipo = factorP.tipo }
factor → **(**bool**)** { factor.tipo = bool.tipo factor.dir = bool.dir }
factor → **numero** { factor.dir = numero.lexval factor.tipo = numero.tipo (lex) }
factor → **cadena** { TablaCadenas.agregar(cadena.lexval) factor.dir =TablaCadenas.getUltimaPos() factor.tipo = cadena }
factor → **true** { factor.dir = 'true' factor.tipo = int } | **false** { factor.dir = 'false' factor.tipo = int }
factorP→ { localizacion.base = factorP.base factorP.dir = nuevaTemporal() } localización { factorP.tipo = localización.tipo genCod(factorP.dir '=' factorP.base '[' localizacion.dir']') }

factorP → ε {
 	
factorP.dir =factorP.base
Si PilaTS.top().buscar(factorP.base) Entonces
factorP.tipo=PilaTS.top().getTipo(factorP.dir)
Sino
factorP.tipo=PilaTS.fondo().getTipo(factorP.dir)
FinSi
}

factorP→ **(**parametros**)** { Si PilaTS.fondo().buscar(factorP.base) Entonces Si PilaTS.fondo().getVar(factorP.base) = 'func' Entonces Si equivalenteListas(PilaTS.fondo().getArgs(factorP.base),parametros.lista) Entonces factorP.tipo = PilaTS.top().getTipo(id) factorP.dir = nuevaTemporal() genCod(factor.dir '=' 'call' id ',' parametros.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  }
```

```
parametros → lista param { parametros.lista = lista_param }

                    | { parametros.lista = NULO }
```

```
lista param →{  lista_paramP.listaHer = nuevaListaArgs()
                    lista_param.listaHer.agregar(bool.tipo)  } lista_paramP
 bool {   lista_param.lista = lista_paramP.listaSin
          genCode(’param’ bool.dir)  }
```

```
lista_paramP→ , bool {  lista_paramP1.listaHer = lista_paramP.listaHer
                            lista_paramP1.listaHer.agregar (bool.tipo)
                           genCode(’param’ bool.dir)}
  lista_paramP1 { lista_paramP.listaSin = lista_paramP1.listaSin }

 | { lista_paramP.listaSin = lista_paramP.listaHer }
```

```
localización→[ bool ] {
Si PilaTS.top().buscar(localizacion.base) Entonces
     Si bool.tipo = int Entonces
        tipoTmp = PilaTS.top().getTipo(localizacion.base)
        Si PilaTT.top().getNombre(tipoTmp)=’array’ Entonces
            localizacionP.tipo = PilaTT.top().getTipoBase(tipoTmp)
            localizacionP.dir = nuevaTemporal()
            localizacionP.tam = PilaTT.top().getTam(localizacionP.tipo)
          genCod(localizacion.dir ’=’ bool.dir ’*’ localizacion.tam )
       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
localizacionP {   localizacion.dir = localizacionP.dirS
                  localizacion.tipo = localizacionP.tipoS  }
```

localizacionP →**[** bool **] {** Si bool.tipo = int Entonces Si PilaTT.top.getNombre(localizacionP.tipo)=”array” Entonces localizacionP$_1$.tipo = PilaTT.top().getTipoBase(localizacionP.tipo) dirTmp = nuevaTemporal() localizacionP$_1$.dir = nuevaTemporal() localizacionP$_1$.tam = PilaTT.top().getTam(localizacionP.tipo) } localizacionP1 { genCod(dirTmp’=’bool.dir’*’ localizacionP$_1$.tam genCod(localizacionP$_1$.dir ’=’localizacionP.dir’+’dirTmp) localizacionP.dirS = localizacionP$_1$.dirS localizacionP.tipoS = localizacionP$_1$.tipoS Sino error(”El id no es un arreglo”) FinSi Sino error(”El índice del arreglo debe ser entero”) FinSi }
localizacionP → ε { localizacionP.dirS=localizacionP.dir localizacionP.tipoS=localizacion.tipo }

Referencias.

- Anónimo. (s.f.) *Traducción dirigida por la sintaxis.* Recuperado el 02 de Febrero de 2021, de http://www2.ulpgc.es/hege/almacen/download/36/36903/capitulo_5.pdf

- Aho, Alfred. (2008). *Compiladores, principios, técnicas y herramientas.* 2a edición. Pearson.