$programa \rightarrow declaraciones funciones$	STS.push(nuevaTS())
	STS.push(nuevaTS())
	dir = 0
	programa.code = funciones.code
$declaraciones \rightarrow tipo\ lista_var$; $declaraciones$	Tipo = tipo.tipo
declaraciones	Tipo = tipo_registro.tipo
→ tipo_registro lista_var ; declaraciones	
$declaraciones \rightarrow \varepsilon$	declaraciones.tipo = base
$tipo_registro \rightarrow estructura inicio declaraciones fin$	STS.push(nuevaTS())
	STT.push(nuevaTT())
	Sdir. push(dir)
	dir = 0 $dir = Sdir.pop()$
	Ts = STS.pop()
	Tt = STT.pop()
	ts.tt = Tt
	T.tipo = STT.getCima().append('struct',tam,Ts)
$tipo \rightarrow base tipo_arreglo$	Base = base.base
	Tipo.tipo = tipo_arreglo.tipo
$base \rightarrow ent$	base.base = ent
$base \rightarrow real$	base.base = real
$base \rightarrow dreal$	base.base = dreal
$base \rightarrow car$	base.base = car
$base \rightarrow sin$	base.base = sin
$tipo_arreglo \rightarrow (num) tipo_arreglo_1$	Si num. tipo = ent Entonces
	Si num. dir > 0 Entonces tipo_arreglo. tipo
	$= TT. append("arreglo", num. dir, tipo_arreglo_1. tipo)$
	Sino
	Error("El valor debe ser positivo")
	Fin Si
	Sino
	Error("El valor debe ser entero")
	Fin Si
$tipo_arreglo \rightarrow \varepsilon$	tipo_arreglo = base
$lista_var \rightarrow lista_var_1$, id	Si!Ts.existe(id) Entonces
	STS. getCima(). append(id, dir, Tipo, 'var', nulo, -1) $Dir \leftarrow dir + STT. getCima(). getTam(Tipo)$
	Sino
	Error("El id ya fue declarado anteriormente")
	Fin Si
$lista_var \rightarrow id$	Si!Ts.existe(id) Entonces
	STS.getCima().append(id,dir,Tipo,'var',nulo,-1)
	$Dir \leftarrow dir + STT.getCima().getTam(Tipo)$
	Sino
	Error("El id ya fue declarado anteriormente")
	Fin Si
funciones y time id/au aumento Nivisis	CilCTC gatCimg() avigta(id) Fatanasa
funciones → tipo id(argumentos)inicio declaraciones sentencias fin funciones	Si!STS.getCima().existe(id) Entonces STS.push(nuevaTS())
decial actories semencias j in j uniciones	Sdir. push(dir)
	dir = 0
	lista_retorno = nuevaLista()
	Si cmpRet(lista_retorno, T. tipo) Entonces
	L = nuevaEtiqueta()
	backpatch(S.nextlist, L)
	F.code = etiqueta(id) S.code etiqueta(L)
	Sino
	Error("El valor no corresponde al tipo de la funcion")
	Fin Si

Sino Sino Error("El id ya fue declarado") Fin SI argumentos → lista_arg argumentos . lista = lista_arg. lista argumentos → sin argumentos . mam = lista_arg. mam argumentos . mam = lista_arg. lista argumentos . mam = lista_arg. lista lista_arg. lista = lista arg., lista lista_arg. lista append(arg. Tipo) lista_arg. nam = lista arg. nam + 1 arg → tipo arg id arg. tipo = lista_arg. mam + 1 arg → tipo arg id arg. tipo = lipo arg. tipo tipo_arg. → base param_arr Base = base. tipo tipo_arg. → base param_arr param. arr. → param. arr. tipo = STT. append("array", -, param_arr, tipo) param. arr. → e sentencias → sentencias, sentencia sentencias → sentencias, sentencia sentencias → sentencias, sentencia sentencias → sentencia sentencias → sentencia sentencias → sentencia sentencias - sentencia sentencias - sentencia sentencias - sentencia sentencia - sentencia sentencia - sentencia, isno sentencia, fin sentencia. code letiqueta(L) sentencia, code la = nuevaEtiqueta() backpatch(e_bool. truelist, L) sentencia. code letiqueta(L) sentencia, code la = nuevaEtiqueta() backpatch(e_bool. falselist, l_) sentencia. nextlist = combinar(e_bool. falselist, sentencia, nextlist) sentencia. nextlist = combinar(sentencia, nextlist) lackpatch(e_bool. falselist, l_) sentencia, nextlist(l0) letiqueta(L_) lackpatch(e_bool. truelist, L_) backpatch(e_bool. truelist, L_) backpatch(e_bool. truelist, L_) backpatch(e_bool. truelist, L_) backpatch(e_bool. falselist, l_) sentencia, nextlist(l0) letiqueta(L_) sentencia, nextlist(l0) letiqueta(L_) sentencia, nextlist = letiqueta(L_) sentencia, nextlist sentencia, code letiqueta(L_) sentencia, nextlist sentencia, code letiqueta(L_) sentencia, nextlist sentencia, code letiqueta(L_) sen		CTTC ()
Sino Fin Si argumentos → lista_arg argumentos. lista = lista_arg_lista argumentos. num = lista arg_num argumentos → sin argumentos. num = lista arg_num argumentos. num = lista_arg_lista argumentos. num = 0 lista_arg → lista_arg_, lista lista_arg_lista = lista_arg_, lista lista_arg_lista append(arg_fipo) lista_arg_mum = lista_arg_num + 1 arg_tipo_arg_lista lista_arg_mum = lista_arg_num + 1 lista_arg_mum = lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista_arg_lista lista_arg_lis		STS.pop()
Fin.Si argumentos → lista_arg argumentos → sin lista_arg → lista — sita_arg , lista lista_arg , lista = lista_arg , lista		
argumentos → lista_arg argumentos. → lista_arg argumentos. → lista_arg, lista argumentos. → lista_arg, lista argumentos. → lista_arg, lista argumentos. → lista_arg, lista argumentos. \(\text{lista} = \text{lista} \) arg, \(\text{lista} = \text{lista} \)		
argumentos → lista_arg argumentos.num = lista_arg.num argumentos → sin argumentos.num = lista_arg.num argumentos → sin lista_arg → lista_arg_1, arg lista_arg → lista_arg_1, arg lista_arg , lista = lista_arg_1, lista lista_arg_lista = lista_arg_n, lista lista_arg_lista = lista_arg_n, lista lista_arg_num = lista_arg_num + 1 lista_arg → arg lista_arg_lista = lista_arg_num + 1 lista_arg_lista = lista_arg_num + 1 lista_arg_num = lista_arg_num + 1 lista_arg_lista = lista_arg_num + 1 lista_arg_lista = lista_arg_lista lista_arg_num + 1 lista_arg_lista = lista_arg_lista lista_arg_lista lista_arg_lista lista_arg_num + 1 lista_arg_lista = lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista lista_arg_lista_arg_lista_arg_lista lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_arg_lista_		
argumentos → sin argumentos .nim = lista_arg_num argumentos .nim = 0 lista_arg → lista_arg_1, arg lista_arg .lista = nulo lista_arg .lista apg.lista = lista_arg_1.lista lista_arg .lista apg.lista lista_arg_num = lista_arg_num + 1 lista_arg → arg lista_arg_num = lista_arg_num + 1 lista_arg .num + lista_arg_num + 1 lista_arg .num + lista_arg_num + 1 lista_arg .num + lista_arg_num + 1 arg → tipo_arg id arg_tipo = tipo arg_tipo tipo_arg → base param_arr Base = base.tipo tipo_arg .tipo = param_arr.tipo param_arr → ()param_arr_1 param_arr + tipo = STT.append("array", -, param_arr_1.tipo) param_arr → ε param_arr.tipo = Base L = nuevaltiqueta() backpatch(sentencias, nextlist = sentencia nextlist sentencia .code sentencias → sentencia sentencias → sentencia sentencias - sentencia sentencia fin L = nuevaltiqueta() backpatch(e bool.truelist, L) sentencia nextlist = combinar(e bool. falselist, sentencia, nextlist) sentencia - si e_bool entonces sentencia_1 fin l = nuevaltiqueta() backpatch(e bool.truelist, L) sentencia nextlist = combinar(e bool.falselist, sentencia, nextlist) sentencia - code = e_bool.code etiqueta(L) sentencia, code l = nuevaltiqueta() backpatch(e bool.truelist, L) sentencia nextlist = combinar(sentencia_1, nextlist, sentencia_1, code and		1 1 1 1
argumentos → sin argumentos .lista = nulo argumentos .num = 0 lista_arg → lista_arg1, arg lista_arg .lista = lista_arg1, lista lista_arg .lista = lista_arg1, lista lista_arg .lista = lista_arg1, lista lista_arg .num = lista_arg .num + 1 lista_arg .min = lista_arg .num + 1 lista_arg .lista = lista_arg1. lista lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .lista lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .lista = lista_arg .num + 1 lista_arg .lista = lista_arg .lista = lista_arg .num + 1 lista_arg .num = lista_arg .num + 1 lista_arg .lista = lista_arg .lista = lista_arg .lista = lista_arg .num + 1 lista_arg .num = lista_arg .num + 1 lista_arg .lista = lista_arg .li	$argumentos \rightarrow usta_arg$	
argumentos.num = 0 lista_arg → lista_arg_1, arg		
$ lista_arg_lista_arg_1 lista_arg_lista l$	$argumentos \rightarrow sin$	•
$ lista_arg_num = lista_arg_num + 1 \\ lista_arg_num = lista_arg_num + 1 \\ lista_arg_num = lista_arg_num + 1 \\ lista_arg_lista = lista_arg_num + 1 \\ lista_arg_num = lista_arg_num + 1 \\ lista_arg_num = lista_arg_num + 1 \\ arg_tipo_arg_tipo_arg_tipo_tipo_tipo_arg_tipo_tipo_tipo_tipo_tipo_arg_tipo_tipo_tipo_tipo_tipo_tipo_tipo_tipo$	11.	
$ lista_arg_arg_mm = lista_arg_mm = lista_arg_mm + 1 \\ lista_arg_lista_arg_lista_arg_mm + 1 \\ lista_arg_lista_append(arg_Tipo) \\ lista_arg_mm = lista_arg_mm + 1 \\ arg_tipo = tipo_arg_mm + 1 \\ arg_tip$	$lista_arg ightarrow lista_arg_1$, arg	
$\begin{array}{ll} lista_arg \rightarrow arg & lista_arg.lista = lista_arg.lista \\ lista_arg.lista = append(arg.Tipo) \\ lista_arg.num = lista_arg.num + 1 \\ arg \rightarrow tipo_arg id & arg.tipo = tipo_arg.tipo \\ tipo_arg \rightarrow base param_arr & Base = base.tipo \\ tipo_arg.tipo = param_arr.tipo \\ param_arr \rightarrow ()param_arr, & param_arr.tipo = STT.append("array", -, param_arr, tipo) \\ param_arr \rightarrow \varepsilon & param_arr.tipo = Base \\ sentencias \rightarrow sentencias_1 sentencia & L = nuevaEtiqueta() \\ backpatch(sentencias_n.nextlist_l) \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.sentencia.nextlist \\ sentencias.code = sentencia.code \\ sentencia \rightarrow si\ e_bool\ entonces\ sentencia_1\ fin \\ sentencia \\ \rightarrow si\ e_bool\ entonces\ sentencia_1\ sino\ sentencia_2\ fin \\ backpatch(e_bool\ truelist, L) \\ sentencia\ nextlist \\ = combinar(e_bool\ falselist, sentencia_1\ code \\ letiqueta(L)\ \ sentencia_1\ code $		
$\begin{array}{c} lista_arg.lista_arg.lista_arg.num = lista_arg.num + 1 \\ arg \rightarrow tipo_arg id \\ tipo_arg \rightarrow base param_arr \\ Base = base.tipo \\ tipo_arg \rightarrow base param_arr \\ Base = base.tipo \\ tipo_arg.tipo = param_arr.tipo \\ param_arr \rightarrow ()param_arr_1 \\ param_arr \rightarrow ()param_arr_2 \\ param_arr \rightarrow ()param_arr_3 \\ param_arr \rightarrow ()param_arr_2 \\ param_arr \rightarrow ()param_arr_3 \\ param_arr_1 tipo = Base \\ sentencias \rightarrow sentencias_1 sentencia \\ L = nuevaEtiqueta() \\ backpatch(sentencias_1, nextlist_L) \\ sentencias.nextlist = sentencia.nextlist_1 \\ sentencias.nextlist = sentencia.nextlist_2 \\ sentencias.nextlist = sentencia.nextlist_3 \\ sentencias.code = sentencia.nextlist_4 \\ sentencias.code = sentencia.code_4 \\ sentencias.code = sentencia.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_4 \\ sentencias.code = sentencia_2.nextlist_4 \\ sentencias.code = sentencia_2.nextlist_4 \\ sentencias.code = sentencia_2.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_5 \\ sentencias.code = sentencia_1.nextlist_5 \\ sentencias.code = sentencia_1.nextlist_4 \\ sentencias.code = sentencia_1.nextlist_5 \\ sentencias.code = sentencia_1.nextlist_5 \\ sentencias.code = sentencia_1.nextlist_5 \\ sentencias.code = sentencias.nextlist_5 \\ se$	7' .	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	$lista_arg \rightarrow arg$	
$ \begin{array}{c} arg \rightarrow tipo_arg.id \\ tipo_arg \rightarrow base param_arr \\ pase = base.tipo \\ tipo_arg.tipo = param_arr.tipo \\ param_arr \rightarrow ()param_arr_1 \\ param_arr.tipo = STT.append("array", -, param_arr_1.tipo) \\ param_arr \rightarrow \varepsilon \\ sentencias \rightarrow sentencias_1 sentencia \\ sentencias \rightarrow sentencias_1 sentencia \\ sentencias. \rightarrow sentencias_1 sentencia \\ sentencias sentencia sentencia \\ sentencias sentencia sentencia \\ sentencias sentencia sentencia \\ sentencias sentencia se$		
$ base param_arr \\ base = base.tipo \\ tipo_arg_tipo = param_arr.tipo \\ param_arr \to ()param_arr_1 \\ param_arr tipo = STT.append("array",-,param_arr_tipo) \\ param_arr \to \varepsilon \\ sentencias \to sentencias_1 sentencia \\ sentencias.code \\ = sentencias.code letiqueta(L) lsentencia.code \\ sentencias \to sentencia \\ sentencias.code = sentencia.nextlist \\ sentencias.code = sentencia.code \\ sentencias.code = sentencia.code \\ sentencias.code = sentencia.code letiqueta(L) lsentencia.code \\ sentencia \to sie_bool entonces sentencia_1 fin \\ backpatch(e_bool.truelist,L) \\ sentencia.code = e_bool.code letiqueta(L) lsentencia_1.code \\ l_1 = nuevaEtiqueta() \\ backpatch(e_bool.truelist,L_1) \\ backpatch(e_bool.truelist,L_1) \\ backpatch(e_bool.truelist,L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_bool.code letiqueta(L_1) lsentencia_1.code \\ lgen('goto'sentencia_1.nextlist, sentencia_2.code \\ lgen('goto'sentencia_1.nextlist,L_1) \\ backpatch(e_bool.truelist,L_2) \\ sentencia.nextlist = lantencia_2.code letiqueta(L_2) lsentencia_2.code letiqueta(L_2) lsentencia_2.code letiqueta(L_2) lsentencia_2.code letiqueta(L_2) lsentencia_2.code letiqueta(L_2) lsentencia_2.code lsentencia.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsentencia_2.code lsen$		
$param_arr \rightarrow ()param_arr_1 & tipo_arg.tipo = param_arr.tipo \\ param_arr \rightarrow \varepsilon & param_arr.tipo = Base \\ sentencias \rightarrow sentencias_1 sentencia & L = nuevaEtiqueta(1) \\ backpatch(sentencias_1, nextlist, L) \\ sentencias.code \\ = sentencias.code etiqueta(L) sentencia.code \\ sentencias \rightarrow sentencia \\ sentencias \rightarrow sentencia \\ sentencias \cdot sentencia \\ sentencia sentencia $		
$\begin{array}{lll} param_arr \rightarrow ()param_arr_1 & param_arr.tipo = STT.append("array", -, param_arr_1.tipo) \\ param_arr \rightarrow \varepsilon & param_arr.tipo = Base \\ sentencias \rightarrow sentencias_1 sentencia & L = nuevaEtiqueta() \\ backpatch(sentencias_1.nextlist, L) \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.code & = sentencias.code etiqueta(L) sentencia.code \\ sentencias \rightarrow sentencia & sentencias.nextlist = sentencia.nextlist \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.code = sentencia.code \\ sentencia \rightarrow si\ e_bool\ entonces\ sentencia_1\ fin \\ sentencia.nextlist & = combinar(e_bool.\ falselist, sentencia_1.nextlist) \\ sentencia.nextlist & = combinar(e_bool.\ falselist, sentencia_1.nextlist) \\ sentencia.nextlist & = combinar(sentencia_1.nextlist) \\ sentencia.nextlist & = combinar(sentencia_1.nextlist, L_1) \\ backpatch(sentencia_1.nextlist, L$	$tipo_arg \rightarrow base\ param_arr$	·
$\begin{array}{lll} param_arr \rightarrow \varepsilon & param_arr.tipo = Base \\ sentencias \rightarrow sentencias_1 sentencia & L = muevaEtiqueta() \\ sentencias_1.nextlist = sentencia.nextlist \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.code & = sentencias_1.code etiqueta(L) sentencia.code \\ sentencias \rightarrow sentencia & sentencias_1.code etiqueta(L) sentencia.code \\ sentencia \rightarrow sie_bool entonces sentencia_1 fin & L = nuevaEtiqueta() \\ sentencia \rightarrow sie_bool entonces sentencia_1 fin & L = nuevaEtiqueta() \\ sentencia & combinar(e_bool.falselist, sentencia_1.nextlist) \\ sentencia.code = e_bool.code etiqueta(L) sentencia_1.code \\ L_1 = nuevaEtiqueta() \\ backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_{bool}.code etiqueta(L_1) sentencia_1.code \\ gen('goto' sentencia_1.nextlist(0)) etiqueta(L_2) \\ sentencia_2.code \\ \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.nextlist = e_bool.code etiqueta(L_2) \\ \\ \end{array}$		tipo_arg.tipo = param_arr.tipo
	$param_arr \rightarrow ()param_arr_1$	
$backpatch(sentencias_1.nextlist, L)\\ sentencias. nextlist = sentencia. nextlist\\ sentencias. code\\ = sentencias_1.code etiqueta(L) sentencia. code\\ sentencias \rightarrow sentencia\\ sentencias. nextlist = sentencia. nextlist\\ sentencias. code = sentencia. nextlist\\ sentencias. code = sentencia. code\\ \\ sentencia \rightarrow si e_bool entonces sentencia_1 fin\\ backpatch(e_bool. truelist, L)\\ sentencia. nextlist\\ = combinar(e_bool. falselist, sentencia_1. nextlist)\\ sentencia. code = e_bool. code etiqueta(L) sentencia_1. code\\ \\ L_1 = nuevaEtiqueta()\\ \\ L_2 = nuevaEtiqueta()\\ \\ backpatch(e_bool. truelist, L_1)\\ backpatch(e_bool. truelist, L_2)\\ sentencia. nextlist\\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist)\\ sentencia. code = e_bool. code etiqueta(L_1) sentencia_1. code\\ gen('goto' sentencia_1. nextlist[0]) etiqueta(L_2)\\ sentencia_2. code\\ \\ sentencia_2. code\\ \\ sentencia_1. nextlist, L_1)\\ backpatch(e_bool. truelist, L_2)\\ sentencia. nextlist = e_bool. falselist\\ sentencia. nextlist = e_bool. code etiqueta(L_2) $		
$sentencias.nextlist = sentencia.nextlist \\ sentencias.code \\ = sentencias_1.code etiqueta(L) sentencia.code \\ sentencias \rightarrow sentencia \\ sentencias.nextlist = sentencia.nextlist \\ sentencias.code = sentencia.code \\ sentencia \rightarrow si e_bool entonces sentencia_1 fin \\ L = nuevaEriqueta() \\ backpatch(e_bool.truelist, L) \\ sentencia.nextlist \\ = combinar(e_bool.falselist, sentencia_1.nextlist) \\ sentencia.code = e_bool.code etiqueta(L) sentencia_1.code \\ L_1 = nuevaEtiqueta() \\ backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_{bool}.code etiqueta(L_1) sentencia_1.code \\ gen('goto'sentencia_1.nextlist, sentencia_2.code) \\ sentencia_2.code \\ sentenci$	$sentencias \rightarrow sentencias_1 sentencia$	
$sentencias.code \\ = sentencias.code etiqueta(L) sentencia.code \\ sentencias > sentencia.s.nextlist = sentencia.nextlist \\ sentencias.code = sentencia.code \\ sentencia > si e_bool entonces sentencia_1 fin \\ L = nuevaEriqueta() \\ backpatch(e_bool.truelist, L) \\ sentencia.nextlist \\ = combinar(e_bool.falselist, sentencia_1.nextlist) \\ sentencia.code = e_bool.code etiqueta(L) sentencia_1.code \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_bool.code letiqueta(L_1) sentencia_1.code \\ gen('goto' sentencia_1.nextlist, sentencia_2.code) \\ sentencia > mientras e_bool hacer sentencia_1 fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(sentencia_1.nextlist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$sentencias \rightarrow sentencia \\ sentencias \rightarrow sentencia \\ sentencias \rightarrow sentencia . nextlist \\ sentencias . nextlist \\ sentencias . code \\ sentencias . code \\ sentencias . code = sentencia . code \\ \\ sentencia \rightarrow si e_bool entonces sentencia_1 fin \\ L = nuevaEriqueta(1) \\ backpatch(e_bool. truelist, L) \\ sentencia . nextlist \\ = combinar(e_bool. falselist, sentencia_1 . nextlist) \\ sentencia . code = e_bool. code etiqueta(L) sentencia_1 . code \\ L_1 = nuevaEtiqueta(1) \\ L_2 = nuevaEtiqueta(1) \\ backpatch(e_bool. truelist, L_1) \\ backpatch(e_bool. truelist, L_2) \\ sentencia . nextlist \\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist) \\ sentencia. code = e_bool. code etiqueta(L_1) sentencia_1. code \\ gen('goto' sentencia_1. nextlist[0]) etiqueta(L_2) \\ sentencia \rightarrow mientras e_bool hacer sentencia_1 fin \\ L_1 = nuevaEtiqueta(1) \\ L_2 = nuevaEtiqueta(1) \\ backpatch(sentencia_1. nextlist, L_1) \\ backpatch(sentencia_1. nextlist, L_2) \\ sentencia. nextlist = e_bool. falselist \\ sentencia. code = etiqueta(L_1) e_bool. code etiqueta(L_2) $		
$sentencias \rightarrow sentencia. nextlist = sentencia. nextlist \\ sentencias. code = sentencia. code \\ \\ sentencia \rightarrow si \ e_bool \ entonces \ sentencia_1 \ fin \\ backpatch(e_bool.truelist, L) \\ sentencia. nextlist \\ = combinar(e_bool.falselist, sentencia_1. nextlist) \\ sentencia. code = e_bool. code \ \ etiqueta(L)\ \ sentencia_1. code \\ \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia. nextlist \\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist) \\ sentencia. code = e_bool. code \ \ etiqueta(L_1)\ \ sentencia_1. code \\ \ gen('goto'sentencia_1. nextlist[0])\ \ etiqueta(L_2) \\ \ sentencia \rightarrow mientras \ e_bool\ hacer\ sentencia_1 fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1. nextlist, L_1) \\ backpatch(sentencia_1. nextlist, L_2) \\ sentencia. nextlist = e_bool.falselist \\ sentencia. nextlist = e_bool.falselist \\ sentencia. nextlist = e_bool.code\ \ etiqueta(L_2)\ \ $		
$sentencia \rightarrow si \ e_bool \ entonces \ sentencia_1 \ fin \\ sentencia \rightarrow si \ e_bool \ entonces \ sentencia_1 \ fin \\ backpatch(e_bool. \ truelist, L) \\ sentencia. \ nextlist \\ = combinar(e_bool. \ falselist, sentencia_1. nextlist) \\ sentencia. code = e_bool. code \ etiqueta(L) \ sentencia_1. code \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool. \ truelist, L_1) \\ backpatch(e_bool. \ falselist, L_2) \\ sentencia. \ nextlist \\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist) \\ sentencia. nextlist \\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist) \\ sentencia. \ code = e_{bool}. \ code \ \ tiqueta(L_1) sentencia_1. \ code \\ \ gen('goto'sentencia_1. nextlist[0]) \ \ etiqueta(L_2) \\ \ sentencia. \ nextlist, L_1) \\ backpatch(e_bool. \ truelist, L_2) \\ sentencia. \ nextlist = e_bool. \ falselist \\ sentencia. \ nextlist = e_bool. \ code \ \ etiqueta(L_2) \ \ etiqueta(L$		
$Sentencia \rightarrow si \ e_bool \ entonces \ sentencia_1 \ fin \\ backpatch(e_bool. \ truelist, L) \\ sentencia. nextlist \\ = combinar(e_bool. falselist, sentencia_1. nextlist) \\ sentencia. code = e_bool. code \ \ etiqueta(L) \ \ sentencia_1. code \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool. \ truelist, L_1) \\ backpatch(e_bool. \ truelist, L_2) \\ sentencia. nextlist \\ = combinar(sentencia_1. nextlist, sentencia_2. nextlist) \\ sentencia. code = e_{bool}. code \ \ etiqueta(L_1) \ sentencia_1. code \\ \ gen('goto' sentencia_1. nextlist[0]) \ \ etiqueta(L_2) \\ \ sentencia_2. code \\ sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool. \ truelist, L_1) \\ backpatch(e_bool. \ truelist, L_2) \\ sentencia. nextlist = e_bool. \ falselist \\ sentencia. nextlist = e_bool. \ falselist \\ sentencia. nextlist = e_bool. \ falselist \\ sentencia. code = etiqueta(L_1) \ \ e_bool. code \ \ etiqueta(L_2) \ $	sentencias → sentencia	
$backpatch(e_bool.truelist,L)\\ sentencia.nextlist\\ = combinar(e_bool.falselist,sentencia_1.nextlist)\\ sentencia.code = e_bool.code etiqueta(L) sentencia_1.code\\ L_1 = nuevaEtiqueta()\\ L_2 = nuevaEtiqueta()\\ backpatch(e_bool.truelist,L_1)\\ backpatch(e_bool.truelist,L_2)\\ sentencia.nextlist\\ = combinar(sentencia_1.nextlist,sentencia_2.nextlist)\\ sentencia.code = e_{bool}.code etiqueta(L_1) sentencia_1.code\\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2)\\ sentencia_2.code\\ sentencia \rightarrow mientras e_bool hacer sentencia_1 fin\\ L_1 = nuevaEtiqueta()\\ L_2 = nuevaEtiqueta()\\ backpatch(sentencia_1.nextlist,L_1)\\ backpatch(e_bool.truelist,L_2)\\ sentencia.nextlist = e_bool.falselist\\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$sentencia.nextlist\\ = combinar(e_bool. falselist, sentencia_1.nextlist)\\ sentencia.code = e_bool.code \mid\mid etiqueta(L)\mid\mid sentencia_1.code\\ \\ L_1 = nuevaEtiqueta()\\ L_2 = nuevaEtiqueta()\\ backpatch(e_bool. truelist, L_1)\\ backpatch(e_bool. falselist, L_2)\\ sentencia.nextlist\\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist)\\ sentencia.code = e_{bool}.code \mid\mid etiqueta(L_1)\mid\mid sentencia_1.code\\ \mid\mid gen('goto'sentencia_1.nextlist[0])\mid\mid etiqueta(L_2)\\ \mid\mid sentencia_2.code\\ \\ \\ L_1 = nuevaEtiqueta()\\ L_2 = nuevaEtiqueta()\\ backpatch(sentencia_1.nextlist, L_1)\\ backpatch(sentencia_1.nextlist, L_1)\\ backpatch(sentencia_1.nextlist, L_2)\\ sentencia.nextlist = e_bool.falselist\\ sentencia.code = etiqueta(L_1)\mid\mid e_bool.code\mid\mid etiqueta(L_2)\mid\mid$	$sentencia \rightarrow si \ e_bool \ entonces \ sentencia_1 \ fin$	
$= combinar(e_bool. falselist, sentencia_1. nextlist)$ $sentencia code = e_bool. code \mid\mid etiqueta(L) \mid\mid sentencia_1. code$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(e_bool. truelist, L_1)$ $backpatch(e_bool. falselist, L_2)$ $sentencia nextlist$ $= combinar(sentencia_1. nextlist, sentencia_2. nextlist)$ $sentencia. code = e_{bool}. code \mid\mid etiqueta(L_1) \mid\mid sentencia_1. code$ $\mid\mid gen('goto' sentencia_1. nextlist[0]) \mid\mid etiqueta(L_2)$ $\mid\mid sentencia_2. code$ $sentencia \rightarrow mientras e_bool hacer sentencia_1 fin$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(sentencia_1. nextlist, L_1)$ $backpatch(sentencia_1. nextlist, L_1)$ $backpatch(e_bool. truelist, L_2)$ $sentencia. nextlist = e_bool. falselist$ $sentencia. code = etiqueta(L_1) \mid\mid e_bool. code \mid\mid etiqueta(L_2) \mid\mid$		
$sentencia.code = e_bool.code \mid etiqueta(L) \mid sentencia\code$ $sentencia$ $\rightarrow si e_bool entonces sentencia_sino sentencia_fin$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(e_bool.truelist, L_1)$ $backpatch(e_bool.falselist, L_2)$ $sentencia.nextlist$ $= combinar(sentencia\nextlist, sentencia\nextlist)$ $sentencia.code = e_{bool}.code \mid etiqueta(L_1) \mid sentencia\code$ $\mid gen('goto'sentencia\nextlist[0]) \mid etiqueta(L_2)$ $\mid sentencia\code$ $sentencia \rightarrow mientras e_bool hacer sentencia_fin$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(sentencia\nextlist, L_1)$ $backpatch(sentencia\nextlist, L_2)$ $sentencia.nextlist = e_bool.falselist$ $sentencia.code = etiqueta(L_1) \mid e_bool.code \mid etiqueta(L_2) \mid $		
$ \begin{array}{lll} sentencia & L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.falselist, L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code \\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2) \\ sentencia_2.code \\ \hline \\ sentencia \rightarrow mientras\ e_bool\ hacer\ sentencia_1\ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) \\ \end{array} $		
$\begin{array}{ll} \rightarrow \textit{si e_bool entonces sentencia}_1 \textit{sino sentencia}_2 \textit{fin} & L_2 = \textit{nuevaEtiqueta}() \\ & \textit{backpatch}(e_bool. \textit{truelist}, L_1) \\ & \textit{backpatch}(e_bool. \textit{falselist}, L_2) \\ & \textit{sentencia. nextlist} \\ & = \textit{combinar}(\textit{sentencia}_1.\textit{nextlist}, \textit{sentencia}_2.\textit{nextlist}) \\ & \textit{sentencia. code} = e_{bool}.\textit{code} \mid \textit{etiqueta}(L_1) \textit{sentencia}_1.\textit{code} \\ & \mid \textit{gen}('\textit{goto'sentencia}_1.\textit{nextlist}[0]) \mid \textit{etiqueta}(L_2) \\ & \mid \textit{sentencia}_2.\textit{code} \\ \\ & \textit{sentencia} \rightarrow \textit{mientras e_bool hacer sentencia}_1 \textit{fin} \\ & L_1 = \textit{nuevaEtiqueta}() \\ & L_2 = \textit{nuevaEtiqueta}() \\ & L_2 = \textit{nuevaEtiqueta}() \\ & \textit{backpatch}(\textit{sentencia}_1.\textit{nextlist}, L_1) \\ & \textit{backpatch}(\textit{sentencia}_1.\textit{nextlist}, L_2) \\ & \textit{sentencia. nextlist} = e_bool. \textit{falselist} \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{sentencia. code} = \textit{etiqueta}(L_1) \mid \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{e_bool. code} \mid \textit{etiqueta}(L_2) \mid \\ \\ & \textit{e_bool. code} \mid \textit{e_tiqueta}(L_2) \mid \\ \\ & \textit{e_bool. code} \mid $		
$backpatch(e_bool.truelist, L_1) \\ backpatch(e_bool.falselist, L_2) \\ sentencia.nextlist \\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code \\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2) \\ sentencia_2.code \\ \\ sentencia \rightarrow mientras e_bool hacer sentencia_1 fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(sentencia_1.nextlist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$backpatch(e_bool.\ falselist, L_2)$ $sentencia.\ nextlist$ $= combinar(sentencia_1.nextlist, sentencia_2.nextlist)$ $sentencia.\ code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code$ $ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2)$ $ sentencia_2.code$ $sentencia \rightarrow mientras\ e_bool\ hacer\ sentencia_1\ fin$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(sentencia_1.nextlist, L_1)$ $backpatch(sentencia_1.nextlist, L_2)$ $sentencia.\ nextlist = e_bool.\ falselist$ $sentencia.\ code = etiqueta(L_1) e_bool.\ code\ \ etiqueta(L_2) $	\rightarrow si e_bool entonces sentencia ₁ sino sentencia ₂ fin	
$sentencia.nextlist\\ = combinar(sentencia_1.nextlist, sentencia_2.nextlist)\\ sentencia.code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code\\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2)\\ sentencia_2.code\\ \\ sentencia \rightarrow mientras\ e_bool\ hacer\ sentencia_1\ fin\\ L_1 = nuevaEtiqueta()\\ L_2 = nuevaEtiqueta()\\ backpatch(sentencia_1.nextlist, L_1)\\ backpatch(sentencia_1.nextlist, L_2)\\ sentencia.nextlist = e_bool.\ falselist\\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$= combinar(sentencia_1.nextlist, sentencia_2.nextlist) \\ sentencia.code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code \\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2) \\ sentencia_2.code \\ \\ sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$sentencia.code = e_{bool}.code \ etiqueta(L_1) sentencia_1.code \\ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2) \\ sentencia_2.code \\ \\ sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$ \ gen('goto'sentencia_1.nextlist[0]) \ \ etiqueta(L_2)$ $ \ sentencia_2.code$ $sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin$ $L_1 = nuevaEtiqueta()$ $L_2 = nuevaEtiqueta()$ $backpatch(sentencia_1.nextlist, L_1)$ $backpatch(sentencia_1.nextlist, L_2)$ $sentencia.nextlist = e_bool.falselist$ $sentencia.code = etiqueta(L_1) \ \ e_bool.code \ \ etiqueta(L_2) \ $		
$ sentencia_2.code \\ sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(sentencia_1.nextlist, L_2) \\ sentencia.nextlist = sebool. \ falselist \\ sentencia.code = stiqueta(L_1) sebool.code stiqueta(L_2) $		$ sentencia.code = e_{bool}.code etiqueta(L_1) sentencia_1.code $
$ \begin{array}{ll} sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) \mid\mid e_bool.code \mid\mid etiqueta(L_2) \mid\mid \\ \end{array} $		$ gen('goto'sentencia_1.nextlist[0]) etiqueta(L_2)$
$ \begin{array}{ll} sentencia \rightarrow mientras \ e_bool \ hacer \ sentencia_1 \ fin \\ L_1 = nuevaEtiqueta() \\ L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) \mid\mid e_bool.code \mid\mid etiqueta(L_2) \mid\mid \\ \end{array} $		sentencia ₂ .code
$L_2 = nuevaEtiqueta() \\ backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $	sentencia → mientras e_bool hacer sentencia₁ fin	
$backpatch(sentencia_1.nextlist, L_1) \\ backpatch(e_bool.truelist, L_2) \\ sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $	•	
$backpatch(e_bool.truelist, L_2)$ $sentencia.nextlist = e_bool.falselist$ $sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$sentencia.nextlist = e_bool.falselist \\ sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $		
$\underline{\underline{\hspace{2cm}}} sentencia_1. code \mid\mid gen('goto'sentencia_1. nextlist[0])$		$sentencia.code = etiqueta(L_1) e_bool.code etiqueta(L_2) $
		$sentencia_1.code gen('goto'sentencia_1.nextlist[0])$
$sentencia \rightarrow hacer sentencia_1 mientras e_bool;$ $L_1 = nuevaEtiqueta()$	$sentencia \rightarrow hacer sentencia_1 mientras e_bool;$	
$L_2 = nuevaEtiqueta()$	·	1 1
$backpatch(sentencia_1.nextlist, L_1)$		
$backpatch(e_bool.truelist, L_2)$		
$sentencia.nextlist = e_bool.falselist$		
$sentencia.code = etiqueta(L_2) sentencia_1.code $		

	$etiqueta(L_1) \mid\mid e_bool.code \mid\mid gen('goto'sentencia_1.nextlist[0])$
sentencia	$L_1 = nuevaEtiqueta()$
→ segun (variable)hacer casos predeterminado fin	$L_2 = nuevaEtiqueta()$
	$backpatch(sentencia.truelist, L_1)$
	backpatch(sentencia. falselist, L ₂)
	sentencia.nextlist
	= combinar(casos.nextlist, predeterminado.nextlist)
	$sentencia.code = variable.code \mid\mid etiqueta(L_1) \mid\mid casos.code \mid\mid$
	$gen('goto' casos.nextlist[0]) etiqueta(L_2) $
	predeterminado, code
$sentencia \rightarrow variable := expresion;$	sentencia.nextlist = null
settletteta - variable - expression,	Si TS. existe(variable) Entonces
	$tipo_variable = TS. getTipo(variable)$
	$t = reducir(expression.dir, expression.tipo, tipo_variable)$
	$t = reductr(expression.utr, expression.utpo, tipo_variable)$ $sentencia.codegen = variable(' = 't)$
	· · · · · · · · · · · · · · · · · · ·
	Sino error(La variable no ha sido declarada)
	Fin Si
$sentencia \rightarrow escribir expresion;$	sentencia.code = gen("printf" expresion.dir)
	sentencia.nextlist = null
$sentencia \rightarrow leer\ variable;$	sentencia.code = gen("scanf" variable.dir)
	sentencia.nextlist = null
$sentencia \rightarrow devolver;$	sentencia.code = gen("return")
	sentencia.nextlist = null
$sentencia \rightarrow devolver\ expression;$	lista_retorno.append(expresion.tipo)
	sentencia.code = gen(return expression.dir)
	sentencia.nextlist = null
$sentencia \rightarrow terminar;$	L = nuevaEtiqueta()
	sentencia.code = gen('goto' L)
	sentencia.nextlist = nuevaLista()
	sentencia.nextlist.add(L)
$sentencia \rightarrow inicio sentencias fin$	sentencia.nextlist = sentencias.nextlist
$casos \rightarrow caso num: sentencia casos_1$	$L_1 = nuevaEtiqueta()$
-	$L_2 = nuevaEtiqueta()$
	$backpatch(num.truelist, L_1)$
	$backpatch(num. falselist, L_2)$
	$casos.nextlist = combiar(sentencia.nextlist, casos_1.nextlist)$
	$casos.code = num.dir etiqueta(L_1) sentencia.code $
	$gen('goto' sentencia.nextlist[0]) etiqueta(L_2) casos_1 .code$
casos → caso num: sentencia	$L_1 = nuevaEtiqueta()$
	$backpatch(num.truelist, L_1)$
	casos.code = num.dir etiqueta(L_1) sentencia.code
	gen('goto' sentencia.nextlist[0])
$predeterminado \rightarrow pred: sentencia$	$L_1 = nuevaEtiqueta()$
predeterminado -> pred. Semencia	$backpatch(num. falselist, L_1)$
	$predeterminado.code = num.dir etiqueta(L_1) $
	sentencia.code gen('goto' sentencia.nextlist[0])
$predeterminado \rightarrow \varepsilon$	predeterminado.code = null
	L = nuevaEtiqueta()
$e_bool \rightarrow e_bool_1 \ o \ e_bool_2$	
	backpatch(e_bool_1.flaselist, L)
	e_bool.truelist = combinar(e_bool ₁ .truelist, e_bool ₂ .truelist)
	$e_bool.falselist = e_bool_2.falselist$
, , , , , ,	$e_bool.code = e_bool_1.code etiqueta(L) e_bool_2.code$
$e_bool \rightarrow e_bool_1 \ y \ e_bool_2$	L = nuevaEtiqueta()

	1 1 (1/ 1 1 (1' (1)
	$backpatch(e_bool_1.truelist, L)$
	$e_bool.truelist = e_bool_2.truelist$
	$e_bool_falselist = combinar(e_bool_1.falselist, e_bool_2.falselist)$
, , , , ,	$e_bool.code = e_bool_1.code etiqueta(L) e_bool_2.code$
$e_bool \rightarrow no \ e_bool_1$	$e_bool_truelist = e_bool_1_falselist$
	$e_bool.falselist = e_bool_1.truelist$
	$e_bool.code = e_bool_1.code$
$e_bool \rightarrow relacional$	$e_bool.truelist = relacional.truelist$
	$e_bool. falselist = relacional. falselist$
$e_bool \rightarrow verdadero$	$t_0 = nuevoIndice()$
	$e_bool.truelist = crearLista(t_0)$
	$e_bool.code = gen('goto' t_0)$
$e_bool \rightarrow falso$	$t_0 = nuevoIndice()$
	$e_bool. falselist = crearLista(t_0)$
	$e_bool.code = gen('goto' t_0)$
$relacional \rightarrow relacional_1 > relacional_2$	relacional.dir = nuevaTemporal
1 2	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_2 = ampliar(relacional_2. dir, relacional_2. tipo, relacional. tipo)$
	relacional. code = $gen(relacional.dir = t'_1 > t'_2)$
$relacional \rightarrow relacional_1 < relacional_2$	relacional.dir = nuevaTemporal
Total Comment (Total Comment)	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_1 = ampliar(relacional_1.dir, relacional_2.tipo, relacional.tipo)$ $t_2 = ampliar(relacional_2.dir, relacional_2.tipo, relacional.tipo)$
	relacional. code = $gen(relacional.dir = t'_1 <' t_2)$
$relacional \rightarrow relacional_1 \le relacional_2$	retactonal. dir = nuevaTemporal
$retuctional \rightarrow retuctional_1 \leftarrow retuctional_2$	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_1 = ampliar(relacional_1, air, relacional_1, tipo, relacional, tipo)$ $t_2 = ampliar(relacional_2, dir, relacional_2, tipo, relacional, tipo)$
	$t_2 = uniplier (relational_2, un', relational_2, tipo, relational, tipo)$ $relational. code = gen(relational, dir = t'_1 <=' t_2)$
malagional malagional >= malagional	
$relacional \rightarrow relacional_1 >= relacional_2$	relacional. dir = nuevaTemporal
	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_2 = ampliar(relacional_2. dir, relacional_2. tipo, relacional. tipo)$
	$relacional.code = gen(relacional.dir = t'_1 >=' t_2)$
$relacional \rightarrow relacional_1 <> relacional_2$	relacional.dir = nuevaTemporal
	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_2 = ampliar(relacional_2. dir, relacional_2. tipo, relacional. tipo)$
	$relacional.code = gen(relacional.dir = t'_1 <>' t_2)$
$relacional \rightarrow relacional_1 = relacional_2$	relacional.dir = nuevaTemporal
	$relacional.tipo = max (relacional_1.tipo, relacional_2.tipo)$
	$t_1 = ampliar(relacional_1. dir, relacional_1. tipo, relacional. tipo)$
	$t_2 = ampliar(relacional_2. dir, relacional_2. tipo, relacional. tipo)$
	$relacional.code = gen(relacional.dir = t'_1 = t'_2)$
$relacional \rightarrow expresion$	relacional. dir = expresion. dir
	relacional.code = expresion.code
$expresion \rightarrow expresion_1 + expresion_2$	expresion.dir = nuevaTemporal
	$expresion.tipo = (expresion_1.tipo, expresion_2.tipo)$
	$t_1 = ampliar(expresion_1.dir, expresion_1.tipo, expresion.tipo)$
	$t_2 = ampliar(expresion_2. dir, expresion_2. tipo, expresion. tipo)$
	$expresion.code = gen(expresion.dir' = t'_1 + t'_2)$
$expresion \rightarrow expresion_1 - expresion_2$	expresion.dir = nuevaTemporal
. 1 1 1 2	$expresion.tipo = (expresion_1.tipo, expresion_2.tipo)$
	The second secon

	$t_1 = ampliar(expression_1. dir, expression_1. tipo, expression. tipo)$
	$t_2 = ampliar(expression_2. dir, expression_2. tipo, expression. tipo)$
	$expresion.code = gen(expresion.dir' = t_1'-t_2)$
$expresion \rightarrow expresion_1 * expresion_2$	expresion.dir = nuevaTemporal
	$expresion.tipo = (expresion_1.tipo, expresion_2.tipo)$
	$t_1 = ampliar(expression_1. dir, expression_1. tipo, expression. tipo)$
	$t_2 = ampliar(expresion_2. dir, expresion_2. tipo, expresion. tipo)$
	$expresion.code = gen(expresion.dir' = t'_1 * t'_2)$
$expression \rightarrow expression_1/expression_2$	expresion.dir = nuevaTemporal
	$expresion.tipo = (expresion_1.tipo, expresion_2.tipo)$
	$t_1 = ampliar(expression_1. dir, expression_1. tipo, expression. tipo)$
	$t_2 = ampliar(expresion_2. dir, expresion_2. tipo, expresion. tipo)$
	$expresion.code = gen(expresion.dir' = t'_1/t_2)$
$expresion \rightarrow expresion_1\% expresion_2$	expresion.dir = nuevaTemporal
	$expresion.tipo = (expresion_1.tipo, expresion_2.tipo)$
	$t_1 = ampliar(expresion_1. dir, expresion_1. tipo, expresion. tipo)$
	$t_2 = ampliar(expresion_2. dir, expresion_2. tipo, expresion. tipo)$
	$expresion.code = gen(expresion.dir' = t'_1\%'t_2)$
expresion → variable	Si TS. existe(variable) Entonces
•	expresion.dir = variable.dir
	expresion.tipo = TS.getTipo(variable)
	Sino
	Error("La variable no ha sido declarada")
	Fin Si
$expresion \rightarrow num$	expresion.tipo = num.tipo
onpression runni	expression.dir = num.val
$expresion \rightarrow cadena$	expresion.tipo = cadena.tipo
expression reduction	expression.tipo = caacha.tipo expression.dir = TablaDeCadenas.add(cadena.val)
$expresion \rightarrow caracter$	expression.tipo = caracter.tipo
expresion / curacter	expression.tipb = curucter.tipb expression.dir = TablaDeCadens.add(caracter.val)
$variable \rightarrow id \ variable_comp$	Si TS. existe(id) Entonces
variable → la variable_comp	$tipo_id = TS. getTipo()$
	$t = reducir(variable_comp.dir,variable_comp.tipo_id)$
	Sino
	Error("El id no ha sido declarado")
	Fin Si
maniable some \ data act sim	
$variable_comp \rightarrow dato_est_sim$	variable_comp.dir = dato_est_sim.dir
assishle some v sure -1-	variable.code = dato_est_sim.code
$variable_comp \rightarrow arreglo$	variable_comp.dir = arreglo.dir
	variable_comp.base = arreglo.base
	variable_comp.tipo = arreglo.tipo
$variable_comp \rightarrow (parametros)$	$variable_comp.lista = parametros.lista$
	variable_comp.num = parametros.num
$dato_est_sim \rightarrow dato_est_sim.id$	Si!TS.existe(id) Entonces
	STS.getFondo().append(id,dir,Tipo)
	$dir \leftarrow dir + STT.getFondo().getTam(Tipo)$
	Sino
	Error("El id no ha sido declarado")
$dato_est_sim \rightarrow \varepsilon$	
$arreglo \rightarrow (expresion)$	t = nuevaTemporal()
	arreglo.dir = nuevaTemporal()
	arreglo.tipo = array
	arreglo.tam = TT.getTam(expresion.tipo)
	arreglo. base = expresion. base
	1 0

	arreglo.code = gen(t' = 'expresion.dir' * 'arreglo.tam)
$arreglo \rightarrow arreglo_1(expresion)$	$Si\ TT.\ getNombre(arreglo_1.tipo) = array\ Entonces$
	t = nuevaTemporal()
	arreglo.dir = nuevaTemporal()
	$arreglo.tipo = TT.getTipoBase(arreglo_1.tipo)$
	$arreglo.tam = TT.getTam(arreglo_1.tipo)$
	$arreglo.base = arreglo_1.base$
	arreglo.code = gen(t' = 'expresion.dir' * 'arreglo.tam)
	$gen(arreglo.dir' = 'arreglo_1.dir' + 't)$
	Sino
	Error("La variable asociada no es un arreglo")
	Fin Si
$parametros \rightarrow lista_param$	$parametros.lista = lista_param.lista$
•	parametros.num = lista_param.num
$parametros \rightarrow \varepsilon$	parametros.lista = null
	parametros.num = 0
$lista_param \rightarrow lista_param_1, expression$	lista_param.lista = nuevaLista()
	lista_param.lista.append(expresion.tipo)
	$lista_param.num = lista_param_1.num + 1$
$lista_param \rightarrow expresion$	lista_param.lista = nuevaLista()
	lista_param.lista.append(expresion.tipo)
	$lista_param.num = 1$
	1000 μ