Universidad Nacional Autónoma de México
Semestre 2020-2
Compiladores
Juárez Aguilar Osmar
Méndez Cabrera Ana Belem
Morales Garcia Luis Angel
Rodríguez Sánchez José Andrés

Definición Dirigida por Sintaxis

(a) Gramática

----*sin: significa sin tipo, car: tipo caracter*-----

1. programa → declaraciones funciones

2. declaraciones → tipo lista_var**;** declaraciones | tipo_registro lista_var**;** declaraciones | ε

3. tipo_registro → **estructura inicio** declaraciones **fin**

4. tipo → base tipo_arreglo

5. base → **ent** | **real** | **dreal** | **car** | **sin**

6. tipo_arreglo → **[num]** tipo_arreglo | ε

7. lista_var → lista_var**, id** | **id**

8. funciones → **def** tipo **id(**argumentos**) inicio** declaraciones sentencias **fin** funciones | ε

9. argumentos → lista_arg | **sin**

10. lista_arg → lista_arg**,** arg | arg

11. arg → tipo_arg **id**

12. tipo_arg → base param_arr

13. param_arr → **[ ]** param_arr | ε

14. sentencias → sentencias sentencia | sentencia

15. sentencia → **si** e_bool **entonces** sentencia **fin**
| **si** e_bool **entonces** sentencia **sino** sentencia **fin**
| **mientras** e_bool **hacer** sentencia **fin**
| **hacer** sentencia **mientras** e_bool**;**
| **segun (**variable**) hacer** casos predeterminado **fin**
| variable **:=** expresion **;**
| **escribir** expresion **;**
| **leer** variable **;**
| **devolver;**
| **devolver** expresion**;**
| **terminar;**
| **inicio** sentencias **fin**

16. casos → **caso num:** sentencia casos | **caso num:** sentencia

17. predeterminado → **pred:** sentencia | ε

18. e_bool → e_bool **o** e_bool | e_bool **y** e_bool | **no** e_bool | **(** e_bool **)** | relacional
| **verdadero** | **falso**

19. relacional → relacional > relacional
    | relacional < relacional
    | relacional <= relacional
    | relacional >= relacional
    | relacional <> relacional
    | relacional = relacional
    | expresion

20. expresion → expresion + expresion

    | expresion − expresion
    | expresion ∗ expresion
    | expresion / expresion
    | expresion **%** expresion | **(**expresion**)**
    | variable | **num** | **cadena** | **carácter**

21. expresion → **id** variable_comp

22. variable_comp → dato_est_sim | arreglo | **(** parametros **)**

23. dato_est_sim → dato_est_sim **.id** | ε

24. arreglo → **[** expresion **]** | arreglo **[** expresion **]**

25. parametros → lista_param | ε

26. lista_param → lista_param**,** expresion | expresion

(b) Definición dirigida por sintaxis

Variables globales

- tipo
- dir

programa→ declaraciones funciones	dir = 0 SSTACK = init_sym_tab_stack() TSTACK = init_type_tab_stack() SYMTAB = init_sym_tab() TYPTAB = init_typ_tab() SSTACK.push_st(SYMTAB) TSTACK.push_st(TYMTAB)
declaraciones→ tipo lista_var; declaraciones	TYP = tipo.tipo
declaraciones→ tipo_registro lista_var**;** declaraciones	TYP = tipo _registro.tipo
tipo_registro→ **estructura inicio** declaraciones **fin**	SYMTAB = init_sym_tab() TYPSYM = init_typ_tab() SYMTAB.push(newTS()) TYPSYM.push(newt())

	dir = 0 dir = Sdir.pop() TSTACK= SYMTAB.pop() SSTACK= TYPSYM.pop() TYPTAB1 = TSTACK.pop_st() SSTACK.getTop().setType(TYPTAB1) SYMTAB1 = SSTACK.pop_st() dir = Sdir.pop() TYP=TSTACK.getTop().append_type (“estructura”,0,TYPTAB1)
tipo→ base tipo_arreglo	base = base.tipo tipo.tipo = tipo_arreglo.tipo
base→ **ent**	base.tipo = ent
base→ **real**	base.tipo = real
base→ **dreal**	base.tipo = dreal
base→ **car**	base.tipo = car
base→ **sin**	base.tipo = sin
tipo_arreglo→ **[num]** tipo_arreglo	Si num.tipo = ent y num.val > 0 entonces tipo_arreglo.tipo = TSTACK.getTop().append_type(“array”, num.val,tipo_arreglo.tipo) sino Error(“El indice tiene que ser entero y mayor a 0”)
tipo_arreglo→ ε	tipo_arreglo.tipo = base
lista_var→ lista_var**, id**	si SSTACK.getTop().getId(id.lexval) = -1 entonces SSTACK.getTop().append_sym(id.lexval,tipo, dir, ”var”) dir = dir + TSTACK.getTop().getTam(tipo) sino Error(”El identificador ya fue declarado”)
lista_var→ **id**	si SSTACK.getTop().getId(id.lexval) = -1 entonces SSTACK.getTop().append_sym(id.lexval,tipo,dir,”var”) dir=dir+TSTACK.getTop().getTam(tipo) sino Error(”El identificador ya fue declarado”)
funciones→**def** tipo **id(**argumentos**) inicio** declaraciones sentencias **fin** funciones	si SSTACK.getTail().getId(id.lexval) = -1 entonces SSTACK.getTop().append_sym(id.lexval, tipo, —, ”def”) Sdir.push(dir) FuncType = tipo.tipo FuncReturn = false dir = 0 TSTACK.push_st (TYPTAB) SSTACK.push_st (SYMTAB) dir = Sdir.pop() add quad(code,0label0, −, −, id.lexval) L = nuevaEtiqueta() backpatch(code, sentencias.next, L)

	add quad(code,0label0, −, −, L) TSTACK.pop_st () SSTACK.pop_st () dir = Sdir.pop() SSTACK.getTop().append_arg(id.lexval, argumentos.lista) si (tipo.tipo = sin) y (FuncReturn = false) entonces Error(la funcíon no tiene valor de retorno) fin si sino Error(”El identificador ya fue declarado”)
argumentos→ lista_arg	argumentos.lista = lista.arg.lista
argumentos → **sin**	argumentos.lista = nulo
lista_arg→ lista_arg**,** arg	lista arg.lista = lista arg1.lista lista arg.lista.add(arg.tipo) lista arg.num = lista arg1.num +1
lista_arg→ arg	lista arg.lista = lista arg1.lista lista arg.lista.add(arg.tipo) lista arg.num = lista arg1.num +1
arg→ tipo_arg **id**	si SSTACK.getTop().getId(id.lexval) = -1 entonces SSTACK.getTop().append_sym(id.lexval, tipo, dir, ”var”) dir = dir + TSTACK.getTop().getTam(tipo) sino Error(”El identificador ya fue declarado”)
tipo_arg→ base param_arr	base = base.tipo tipo_arg = param_arr.tipo
param_arr→ **[ ]** param_arr	param arr.tipo = TSTACK.getTop().append_type(”array”, -, param arr.tipo)
param_arr→ ε	param_arr.tipo = base
sentencias→ sentencias sentencia	L = nuevaEtiqueta() backpatch(code, sentecias.listnext, L) sentencias.listnext = sentencia.listnext
sentencias→ sentencia	sentencias.listnext = sentencia.listnext
sentencia→ **si** e_bool **entonces** sentencia **fin**	L = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L) sentencia.listnext=combinar(e_bool.listfalse,sentencia.listnext)
sentencia→ **si** e_bool **entonces** sentencia **sino** sentencia **fin**	L = nuevaEtiqueta() L1 = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L) backpatch(code, e_bool.listfalse, L1) sentencia.listnext =combinar(e_bool.listfalse,sentencia.listnext)
sentencia→ **mientras** e_bool **hacer** sentencia **fin** ***********	L1 = nuevaEtiqueta() L2 = nuevaEtiqueta() backpatch(sentencia1.listnext, L1)

	backpatch(e-bool.listtrue, L2) sentencia.nextlist = e_bool.listfalse sentencia.code = etiqueta(L1) ||e_bool.code || etiqueta(L2) || sentencia.code || gen('goto' sentencia.listnext[0]) || sentencia1.code
sentencia→ **hacer** sentencia **mientras** e_bool**;**	L = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L) backpatch(code, sentencia.listnext, L1) sentencia.listnext = e_bool.listfalse add quad(code, "label", -, -, L)
sentencia→ **segun (**variable**) hacer** casos predeterminado **fin**	L1= nuevaEtiqueta() L2= nuevaEtiqueta() backpatch(code,sentencia.listtrue,L1) backpatch(code,sentencia.listfalse,L2) sentence.listnext=combiner(casos.listnext, predeterminado.listnext) sentencia.code=variable.code || etiqueta(L1) || casos.code || etiqueta (L2) || predeterminado.code
sentencia→ variable **:=** expresion **;**	α = reducir(expresion.dir, expresion.tipo, variable.tipo) add quad(code, " = ", α, −, variable.base[variable.dir]) sentecia.listnext= nulo
sentencia→ **escribir** expresion **;**	add quad(code, "print", expresion.dir, -,-) sentecia.listnext= nulo
sentencia→ **leer** variable **;**	add quad(code, "scan", -, -,variable.dir) sentecia.listnext= nulo
sentencia→ **devolver;**	si FuncType = sin entonces add quad(code, "return", −, −, −) sino Error("La funcion debe retornar algun valor de tipo " + FuncType) fin sentencia.listnext= nulo
sentencia→ **devolver** expresion**;**	si FuncType = sin entonces α = reducir(expresion.dir, expresion.tipo, FuncType) add quad(code, "return", expresion.dir, −, −) FuncReturn = true else Error("La funcion no puede retornar algun valor de tipo ") fin sentencia.listnext= nulo
sentencia→ **terminar;**	I = newIndex() add quad(code, "goto", -, -, I)

	sentencia.listnext = newList() sentencia.listnext.add(I)
sentencia→ **inicio** sentencias **fin**	sentencia.listnext=sentencias.listnext
casos→ **caso num:** sentencia casos	backpatch(caso) casos.dir=sentencia.dir casos.val=setencia.val
casos→ **caso num:** sentencia	backpatch(casos) casos.dir=casos1.dir+setencias.dir
predeterminado→ **pred:** sentencia	predeterminado.dir=sentencias.dir
predeterminado→ ε	predeterminado.dir=null
e_bool→ e_bool **o** e_bool	L = nuevaEtiqueta() backpatch(code, e_bool1.listfalse, L) e_bool.listtrue=combinar(e_bool1.listtrue,e_bool2.lissttrue) e_bool.listfalse= e_bool2.listfalse add quad(code, ”label”, -, -, L)
e_bool→ e_bool **y** e_bool	L = nuevaEtiqueta() backpatch(code, e_bool1.listtrue, L) e_bool.listtrue =e_bool2.listtrue e_bool.listfalse=combinar(e_bool1.listfalse,e_bool2.lisstfalse) add quad(code, ”label”, -, -, L)
e_bool→ **no** e_bool	e_bool.listtrue = e_bool1.listfalse e_bool.listfalse = e_bool1.listtrue
e_bool→ **(** e_bool **)**	e_bool.listtrue = e_bool1.listtrue e_bool.listfalse = e_bool1.listfalse
e_bool→ relacional	e_bool.listtrue=relacional.listtrue e_bool.listfalse = relacional.listfalse
e_bool→ **verdadero**	I=newIndex() e_bool.listtrue=newList() e_bool.listtrue.add(I) add quad(code, ”goto”, -, -, I) e_bool.listfalse = nulo
e_bool→ **falso**	I = newIndex() e_bool.listtrue = nulo e_bool.listfalse = newList() e_bool.listfalse.add(I) add quad(code, ”goto”, -, -, I)
relacional→ relacional > relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”>”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)

relacional→relacional < relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo= max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)
relacional→ relacional <= relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)
relacional→ relacional >= relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1 = ampliar(relacional1.dir, relacional1.tipo, relacional.tipo) α2 = ampliar(relacional2.dir, relacional2.tipo, relacional.tipo) add quad(code, ”>=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)
relacional→ relacional <> relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<>”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)
relacional→ relacional = relacional	relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1)

	relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)
relacional→ expresion	relacional.tipo=expresion.tipo relacional.dir = expresion.dir
expresion→ expresion **+** expresion	expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”+”,α1 ,α2, expresion.dir)
expresion→ expresion − expresion	expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”–”,α1 ,α2, expresion.dir)
expresion→ expresion ∗ expresion	expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”∗”,α1 ,α2, expresion.dir)
expresion→ expresion / expresion	expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”/”,α1 ,α2, expresion.dir)
expresion→ expresion **%** expresion	expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”%”,α1 ,α2, expresion.dir)
expresion→ **(**expresion**)**	expresion.dir = expresion1.dir expresion.tipo = expresion1.tipo
expresion→ variable	epxreson.dir = newTemp() expresion.tipo = variable.tipo add quad(code, ”∗”, variable.base[variable.dir] , -, expresion.dir)
expresion→ **num**	expresion.tipo = num.tipo expresion.dir = num.val
expresion→ **cadena**	expresion.tipo = cadena expresion.dir = TabCad.add(cadena)
expresion→ **caracter**	expresion.tipo = caracter expresion.dir = TabCad.add(caracter)
variable→ **id** variable_comp	Si SSTACK.getID() existe entonces expression.dir=variable.dir tipo_id=SSTACK.getTipo()

	t=reducir(variable_comp.dir,variable_comp.tipo,tipo.getID) sino Error(“El ID no ha sido declarado) Fin si
variable_comp→ dato_est_sim	variable_comp.dir=dato_est_sim.dir variable.code=dato_est_sim.code
variable_comp→ arreglo	variable_comp.dir = arreglo.dir variable_comp.base = arreglo.base variable_comp.tipo = arreglo.tipo
variable_comp→ **(** parametros **)**	variable_comp.lista=parametros.lista variable_comp.num=parametros.num
dato_est_sim→ dato_est_sim**.id**	Si SSTACK.getTail().getID(id.lexval) = -1 entonces t = SSTACK.getTail().getTipo(id.lexval) t1 = TSTACK.getTail().getTipo(t) si t1 = tipo_registro entonces tipoBase = TSTACK.getTail().getTipoBase(t) si tipoBase.getID(id) = -1 entonces dato_est_sim.tipo = tipoBase.getTipo(id) dato_est_sim.dir = id dato_est_sim.base = dato_est_sim1.base else Error(“El ID no existe en la estructura”) else Error(“El ID no es una estructura”) else Error(“El ID no ha sido declarado”)
dato_est_sim→ ε	dato_est_sim = base
arreglo→ **[** expresion **]**	t=newTemp() arreglo.dir=newTemp() arreglo.tipo=array arreglo.tam=TT.getTam(tipo) arreglo.base=expresion.base arreglo.code=gen(t ‘=’ expresion.dir ‘*’ arreglo.tam)
arreglo→ arreglo **[** expresion **]**	Si TT.getName(arreglo1.tipo)=array entonces t=newTemp() arreglo.dir=newTemp() arreglo.tipo=TT.getTipoBase(arreglo1.tipo) arreglo.tam=TT.getTam(arreglo1.tipo) arreglo.base=arreglo1.base arreglo.code=gen(t‘ =‘ expresion.dir‘ *‘arreglo.tam || gen(arreglo.dir’ =’ arreglo1.dir’ +’t) Sino Error(“La variable asociada no es un arreglo”) Fin si
parametros→ lista_param	parametros.tipos=lista_param.tipos
parametros→ ε	parametros.tipos=nulo
lista_param→ lista_param**,** expresion	lista_param.tipos = lista_param
lista_param→ expresion	lista_param.tipos = expression.tipos

	lista_param.dir = expression.dir

(c) Esquema de traducción

Variables globales

- tipo
- dir

- programa→ declaraciones {dir = 0
  StackTS = init_sym_tab_stack()
- StackTT = init_type_tab_stack()
- SymTAB = init_sym_tab()
- TypTAB = init_typ_tab()
- StackTS.push_st(SymTAB)
- StackTT.push_st(TypTAB)
- TablaCadenas = init_tabla_cadenas()
- }funciones

declaraciones→ tipo lista_var**;** {typeGBL = tipo.tipo} declaraciones

declaraciones→ tipo_registro lista_var**;** {typeGBL = tipo _registro.tipo} declaraciones

tipo→ base tipo_arreglo{base = base.tipo
tipo.tipo = tipo_arreglo.tipo}

base→ **ent** {base.base=StackTT.getTop().getType("ent")}

base→ **real** {base.base = StackTT.getTop().getType("real")}

base→ **dreal**{base.base = StackTT.getTop().getType("dreal")}

base→ **car**{base.base = StackTT.getTop().getType("car")}

base→ **sin**{base.base = StackTT.getTop().getType("sin")}

tipo_arreglo→ **[num]** {Si num.tipo = ent y num.val > 0 entonces
tipo_arreglo.tipo =StackTT.getTop().append_type("array",num.val,tipo_arreglo.tipo)
sino
Error("El indice tiene que ser entero y mayor a 0")
Fin} tipo_arreglo1

base→ **sin**{base.base = StackTT.getTop().getType("sin")}

tipo_arreglo→ **[num]** {Si num.tipo = ent y num.val > 0 entonces
tipo_arreglo.tipo =StackTS.getTop().append_type("array",num.val,tipo_arreglo.tipo)
sino
Error("El indice tiene que ser entero y mayor a 0")
Fin} tipo_arreglo1

tipo_arreglo→ ε{tipo_arreglo.tipo = base}

lista_var→ lista_var**,** **id**{si StackTS.getTop().getId(id.lexval) = -1 entonces
StackTS.getTop().append_sym(id.lexval,tipo, dir,"var")
dir = dir + StackTT.getTop().getTam(tipo)
sino
Error("El identificador ya fue declarado")
fin}

lista_var→ **id**{si StackTS.getTop().getId(id.lexval) = -1 entonces

StackTS.getTop().append_sym(id.lexval,tipo,dir,”var”) dir=dir+ StackTT.getTop().getTam(tipo) sino Error(”El identificador ya fue declarado”) Fin}
funciones→**def** tipo **id**{si StackTS.getTail().getId(id.lexval) = -1entonces StackTS.getTop().append_sym(id.lexval, tipo, −−,”def”) Sdir.push(dir) FuncType = tipo.tipo FuncReturn = false dir = 0 StackTT.push_st (TypTAB) StackTS.push_st (SymTAB) dir = Sdir.pop() add quad(code,0label0, −, −, id.lexval) L = nuevaEtiqueta() backpatch(code, sentencias.next, L) add quad(code,0label0, −, −, L) StackTT pop_st () StackTS.pop_st () dir = Sdir.pop() StackTS.getTop().append_arg(id.lexval,argumentos.lista) si (tipo.tipo = sin) y (FuncReturn = false) entonces Error(la funcíon no tiene valor de retorno) fin si sino Error(”El identificador ya fue declarado”) fin} **(**argumentos**) inicio** declaraciones sentencias **fin** funciones
funciones→ ε {printf(“Fin del programa”); }
argumentos→ lista_arg{argumentos.lista = lista.arg.lista

argumentos.num = lista arg.num}
argumentos → **sin**{argumentos.lista = nulo argumentos.num = 0}
lista_arg→ lista_arg**,** {lista arg.lista = lista arg1.lista lista arg.lista.add(arg.tipo) lista arg.num = lista arg1.num +1 } arg
lista_arg→ arg{lista arg.lista = newListaParam() lista arg.lista.add(arg.tipo) lista arg.num = 1}
arg→ tipo_arg **id**{si StackTS.getTop().getId(id.lexval) = -1 entonces StackTS.getTop().append_sym(id.lexval, tipo, dir,"var") dir = dir + StackTT.getTop().getTam(tipo) sino Error("El identificador ya fue declarado") fin arg.tipo = tipo_arg.tipo}
tipo_arg→ base param_arr{base = base.tipo tipo_arg = param_arr.tipo}
param_arr→ **[ ]** {param arr.tipo = StackTT.getTop().append_type("array",0, param arr.tipo) } param_arr
param_arr→ ε{ param_arr.tipo = base }
sentencias→ sentencias1{L = nuevaEtiqueta() backpatch(code, sentecias.listnext, L) sentencias.listnext = sentencia.listnext} sentencia
sentencias→ sentencia {sentencias.listnext = sentencia.listnext}
sentencia→ **si** e_bool {L = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L)

sentencia.listnext=combinar(e_bool.listfalse,sentencia.listnext) genCode(label L)} **entonces** sentencia1 **fin**
sentencia→ **si** e_bool **entonces** sentencia1 **sino** sentencia2 **fin** {L1 = nuevaEtiqueta() L2 = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L1) backpatch(code, e_bool.listfalse, L2) sentencia.listnext =combinar(e_bool.listfalse,sentencia.listnext) genCode(label L1) genCode('goto' sentencia1.listnext[0]) genCode(label L2)}
sentencia→ **mientras** e_bool {L1 = nuevaEtiqueta() L2 = nuevaEtiqueta() backpatch(sentencia1.listnext, L1) backpatch(e-bool.listtrue, L2) sentencia.nextlist = e_bool.listfalse sentencia.code = etiqueta(L1) ||e_bool.code || etiqueta(L2) || sentencia.code || gen('goto' sentencia.listnext[0]) || sentencia1.code}**hacer** sentencia **fin**
sentencia→ **hacer** sentencia1 {L = nuevaEtiqueta() backpatch(code, e_bool.listtrue, L) backpatch(code, sentencia.listnext, L1) sentencia.listnext = e_bool.listfalse add quad(code, "label", -, -, L)} **mientras** e_bool**;**
sentencia→ **segun (**variable**) hacer** casos predeterminado **fin** {L1= nuevaEtiqueta() L2= nuevaEtiqueta() backpatch(code,sentencia.listtrue,L1) backpatch(code,sentencia.listfalse,L2)

sentence.listnext=combiner(casos.listnext, predeterminado.listnext) sentencia.code=variable.code || etiqueta(L1) || casos.code || etiqueta (L2) || predeterminado.code}
sentencia→ variable **:=** expresion **;** {si StackTS.getCima().getId(id.lexval)= -1 entonces t = StackT S.getCima().getT ipo(id.lexval) d = StackT S.getCima().getDir(id.lexval) α=reducir(expresion.dir,expresion.tipo,t) add quad(code, ” = ”, α, −, ”Id” + d) Sino Error(”El identificador no ha sido declarado”) fin sentecia.listnext= nulo add quad(code, ” = ”, α, −,variable.base[variable.dir]) sentecia.listnext= nulo}
sentencia→ **escribir** expresion **;** {add quad(code, ”print”, expresion.dir, -,-) sentecia.listnext= nulo}
sentencia→ **leer** variable **;** {add quad(code, ”scan”, -, -,variable.dir) sentecia.listnext= nulo}
sentencia→ **devolver;** {si FuncType = sin entonces add quad(code, ”return”, −, −, −) sino Error(”La funcion debe retornar algun valor de tipo ”+ FuncType) fin sentencia.listnext= nulo}
sentencia→ **devolver** expresion**;** {si FuncType = sin entonces α = reducir(expresion.dir, expresion.tipo,FuncType) add quad(code, ”return”, expresion.dir, −, −) FuncReturn = true sino Error(”La funcion no puede retornar algun valor de tipo ”) fin

sentencia.listnext= nulo}
sentencia→ **terminar;** {I = newIndex() add quad(code, ”goto”, -, -, I) sentencia.listnext = newList() sentencia.listnext.add(I)}
sentencia→ **inicio** sentencias1 **fin** { sentencia.listnext=sentencias1.listnext }
casos→ **casos1caso num:**sentencia{casos.nextlist=combinar(casos.nextlist,sentencia1.nextlist) L = newLabel() /*Indica el inicio del codigo para la sentencia*/ backpatch(casos) casos.dir=casos1.dir+setencias.dir genCode(”label” L) casos.prueba=casos1.prueba casos.prueba.append(if ”??” ”==” num.dir ”goto” L)}
casos→ **caso num:** sentencia{casos.prueba = newCode() L = newLabel() backpatch(caso) casos.dir=sentencia.dir casos.val=setencia.val}
predeterminado→ **pred:** sentencia{predeterminado.prueba = newCode() L = newLabel() /*Indica el inicio del c´odigo para la sentencia*/ predeterminado.dir=sentencias.dir genCode(”label”L) predeterminado.prueba.append(”goto” L) predeterminado.dir=sentencias.dir}
predeterminado→ ε{predeterminado.dir=null}
e_bool→ e_bool **o** e_bool{L = nuevaEtiqueta() backpatch(code, e_bool1.listfalse, L) e_bool.listtrue=combinar(e_bool1.listtrue,e_bool2.lissttrue)

e_bool.listfalse= e_bool2.listfalse add quad(code, ”label”, -, -, L)}
e_bool→ e_bool **y** e_bool{L = nuevaEtiqueta() backpatch(code, e_bool1.listtrue, L) e_bool.listtrue =e_bool2.listtrue e_bool.listfalse=combinar(e_bool1.listfalse,e_bool2.lisstfalse) add quad(code, ”label”, -, -, L) genCode(label L)}
e_bool→ **no** e_bool{e_bool.listtrue = e_bool1.listfalse e_bool.listfalse = e_bool1.listtrue}
e_bool→ **(** e_bool **)** {e_bool.listtrue = e_bool1.listtrue e_bool.listfalse = e_bool1.listfalse}
e_bool→ relacional{e_bool.listtrue=relacional.listtrue e_bool.listfalse = relacional.listfalse}
e_bool→ **verdadero**{I=newIndex() e_bool.listtrue=newList(I) e_bool.listtrue.add(I) add quad(code, ”goto”, -, -, I) e_bool.listfalse = nulo genCode(’goto’ index0)}
e_bool→ **falso**{I = newIndex() e_bool.listtrue = nulo e_bool.listfalse = newList(I) e_bool.listfalse.add(I) add quad(code, ”goto”, -, -, I) genCode(’goto’ index0)}
relacional→ relacional > relacional{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I)

	relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”>”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}
relacional→relacional < relacional	{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo= max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}
relacional→ relacional <= relacional	{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}
relacional→ relacional >= relacional	{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex()

relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1 = ampliar(relacional1.dir, relacional1.tipo,relacional.tipo) α2 = ampliar(relacional2.dir, relacional2.tipo,relacional.tipo) add quad(code, ”>=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}
relacional→ relacional <> relacional{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(), I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”<>”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}
relacional→ relacional = relacional{relacional.listtrue = newList() relacional.listfalse = newList() I= newIndex(),I1 = newIndex() relacional.listtrue.add(I) relacional.listfalse.add(I1) relacional.tipo=max(relacional1.tipo, relacional2.tipo) α1=ampliar(relacional1.dir,relacional1.tipo,relacional.tipo) α2=ampliar(relacional2.dir,relacional2.tipo,relacional.tipo) add quad(code, ”=”,α1 ,α2, I) add quad(code, ”goto”, -, -, I1)}

relacional→ expresion{relacional.tipo=expresion.tipo relacional.dir = expresion.dir}
expresion→ expresion + expresion{expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”+”,α1 ,α2, expresion.dir)}
expresion→ expresion − expresion{expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”−”,α1 ,α2, expresion.dir)}
expresion→ expresion ∗ expresion{expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”∗”,α1 ,α2, expresion.dir)}
expresion→ expresion / expresion{expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”/”,α1 ,α2, expresion.dir)}
expresion→ expresion % expresion{expresion.tipo = max(expresion1.tipo,expresion2.tipo) expresion.dir = newTemp() α1=ampliar(expresion1.dir,expresion1.tipo,expresion.tipo) α2=ampliar(expresion2.dir,expresion2.tipo,expresion.tipo) add quad(code, ”%”,α1 ,α2, expresion.dir)}
expresion→ (expresion){expresion.dir = expresion1.dir expresion.tipo = expresion1.tipo }
expresion→ variable{epxreson.dir = newTemp()

expresion.tipo = variable.tipo add quad(code, ”∗”, variable.base[variable.dir] , -, expresion.dir)}
expresion→ **num**{expresion.tipo = num.tipo expresion.dir = num.val}
expresion→ **cadena**{expresion.tipo = cadena expresion.dir = TabCad.add(cadena)}
expresion→ **caracter**{expresion.tipo = carácter expresion.dir = TabCad.add(caracter)}
variable→ **id** variable_comp{Si StackTS.getID() existe entonces expression.dir=variable.dir tipo_id= StackTS.getTipo() t=reducir(variable_comp.dir,variable_comp.tipo,tipo.getID) sino Error(“El ID no ha sido declarado) fin si}
variable_comp→ dato_est_sim{variable_comp.dir=dato_est_sim.dir variable.code=dato_est_sim.code}
variable_comp→ arreglo{variable_comp.dir = arreglo.dir variable_comp.base = arreglo.base variable_comp.tipo = arreglo.tipo}
variable_comp→ **(** parametros **)** {variable_comp.lista=parametros.lista variable_comp.num=parametros.num}
dato_est_sim→ dato_est_sim**.id**{Si StackTS.getTail().getID(id.lexval) = -1 entonces t = StackTS.getTail().getTipo(id.lexval) t1 = StackTT.getTail().getTipo(t) si t1 = tipo_registro entonces tipoBase = StackTT.getTail().getTipoBase(t) si tipoBase.getID(id) = -1 entonces dato_est_sim.tipo = tipoBase.getTipo(id) dato_est_sim.dir = id

dato_est_sim.base = dato_est_sim1.base sino Error(“El ID no existe en la estructura”) sino Error(“El ID no es una estructura”) sino Error(“El ID no ha sido declarado”) fin}
dato_est_sim→ ε{dato_est_sim = base typeTemp= StackTS.getTop().getType(id) Si StackTS.getTop().getName(typeTemp) =’struct’ Entonces dato est sim.estructura= true datoestsim.tabla= StackTS getTop().getTipoBase(typeTemp).tabla dato est sim.des = 0 sino dato est sim.estructura= false dato est sim.type = StackTS getTop().getType(id) fin si dato est sim.code est=false}
arreglo→ **[** expresion **]** {arreglo.type = StackTS.getTop().getType(IDGBL) StackTS.getTop().getName(arreglo1.tipo)=array arreglo.type=STS.getTop().getType(IDGBL) si STT.getTop().getName(arreglo.type)=’array’ entonces si expresion.type = entero entonces typeTemp = StackTS.getTop().getTypeBase(arreglo.type) tam = StackTS getTop().getTam(typeTemp) arreglo.dir = newTemp() genCode(arreglo.dir’=’ expresion.dir ’*’ tam) sino error(...) fin Si

sino error(...) fin Si}
arreglo→ arreglo **[** expresion **]** {si StackTS.getTop().getName(arreglo1.tipo)=array entonces t=newTemp() arreglo.dir=newTemp() arreglo.tipo=TT.getTipoBase(arreglo1.tipo) arreglo.tam=TT.getTam(arreglo1.tipo) arreglo.base=arreglo1.base arreglo.code=gen(t‘ =‘ expresion.dir‘ *‘arreglo.tam‖ gen(arreglo.dir’ =’ arreglo1.dir’ +’t) sino Error(“La variable asociada no es un arreglo”) fin si}
parametros→ lista_param{parametros.lista=lista_param.lista parametros.num = lista param.num}
parametros→ ε{parametros.tipos=nulo parametros.num = 0}
lista_param→ lista_param**,** expresion{lista_param.lista=lista_param1.lista lista param.lista.append(expression.tipo) lista param.num = lista param1 + 1}
lista_param→ expresion{lista_param.tipos = expression.tipos lista_param.dir = expression.dir}