**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**

1. 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

1. 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 |

1. 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 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”)  fin} |
| 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”)  Fin} |
| funciones→**def** tipo **id**{si SSTACK.getTail().getId(id.lexval) = -1entonces  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”)  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 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”)  fin  arg.tipo = tipo\_arg.tipo} |
| tipo\_arg→ base param\_arr{base = base.tipo  tipo\_arg = param\_arr.tipo} |
| param\_arr→ **[ ]** {param arr.tipo =TSTACK.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 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  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=SSTACK.getTop().getType(id)  Si SSTACK.getTop().getName(typeTemp) =’struct’ Entonces  dato est sim.estructura= true  datoestsim.tabla=SSTACKgetTop().getTipoBase(typeTemp).tabla  dato est sim.des = 0  sino  dato est sim.estructura= false  dato est sim.type = SSTACKgetTop().getType(id)  fin si  dato est sim.code est=false} |
| arreglo→ **[** expresion **]** {arreglo.type = SSTACK.getTop().getType(IDGBL)  SSTACK.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 = SSTACK.getTop().getTypeBase(arreglo.type)  tam = SSTACK getTop().getTam(typeTemp)  arreglo.dir = newTemp()  genCode(arreglo.dir’=’ expresion.dir ’\*’ tam)  sino  error(...)  fin Si  sino  error(...)  fin Si} |
| arreglo→ arreglo **[** expresion **]** {si SSTACK.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} |