Carrillo Molina Israel
Martínez Martínez Brayan Eduardo
Pachuca Cortés Santiago Emilio

ESQUEMA DE TRADUCCIÓN.

```
programa → declaraciones funciones {
STS.push(newTS())
STT.push(newTT())
dir = 0
programa.código = funciones.codigo
}
```

```
declaraciones → {typeGBL = tipo.type} tipo lista_var ; declaraciones1
```

```
declaraciones → {typeGBL= tipo registro.type} tipo_registro lista_var ; declaraciones
```

```
declaraciones → ε
```

```
tipo_registro → estructura inicio declaraciones fin {
STS.push(newTS())
STT.push(newTT())
SDir.push(dir)
dir = 0
SymTab = STS.pop()
SymTab.typeTab = STT.pop()
tam = getTam(SymTab)
dir = SDir.pop()
tipo_registro.type = STT.getTop().insert("struct", tam, SymTab)
}
```

```
tipo → base tipo_arreglo {
baseGBL = base.base
tipo.type = tipo_arreglo.type
}
```

```
base → ent {base.base = STT.getTop().getType("ent")}
```

```
base → real {base.base = STT.getTop().getType("real")}
```

```
base → dreal {base.base = STT.getTop().getType("dreal")}
```

```
base → car {base.base = STT.getTop().getType("car")}
```

```
base → sin {base.base = STT.getTop().getType("sin")}
```

```
tipo_arreglo → [num] {
Si num.type = ent Entonces
    Si num.dir > 0 Entonces
        tipo_arreglo.type = STT.getTop().insert('array',num, tipo_arreglo1.tipo)
    Sino
        error("El indice debe ser mayor a 0")
    Fin Si
Sino
    error("El indice debe de ser entero")
```

```
Fin Si
} tipo_arreglo1
```

```
tipo_arreglo → {tipo arreglo.type = baseGBL}
```

```
lista_var → lista_var1, id {
Si no STS.getTop().existe(id) Entonces
   STS.getTop().insert(id, typeGBL, dir, 'var', null, null )
         dir = dir + STT.getTop().getTam(typeGBL)
Sino
   error("El id ya existe")
Fin Si
}
```

```
lista_var → id {
Si no STS.getTop().existe(id) Entonces
   STS.getTop().insert(id, typeGBL, dir, 'var', null, null )
   dir = dir + STT.getTop().getTam(typeGBL)
Sino
   error("El id ya existe")
Fin Si
}
```

```
funciones → def tipo id(argumentos) inicio declaraciones sentencias fin funciones
{
Si no STS.getGlobal().existe(id) Entonces
   STS.push(newTS())
   STT.push(newTT())
   SDir.push(dir)
   dir = 0
   listaRET = newListRet()
  Si cmpRet(lista retorno, tipo.type) Entonces
      L = newLabel()
      backpatch(sentencias.nextlist, L)
      genCode(label L)
      STS.pop()
      STT.pop()
  Sino
      error("El valor no corresponde al tipo de la función")
  Fin Si
Sino
  error("El id ya fue declarado")
Fin Si
}
```

```
funciones → ε
```

```
argumentos → lista_arg {
argumentos.lista = lista arg.lista
argumentos.num = lista arg.num
}
```

```
argumentos → sin {
argumentos.lista = NULO
```

argumentos.num = 0
}

lista_arg → $lista\_arg_1$, arg {
lista arg.lista = lista arg1.lista
lista arg.lista.append(arg.type)
lista arg.num = lista arg1.num + 1
}

lista_arg → arg {
lista arg.lista = newList()
lista arg.lista.append(arg.type)
lista arg.num = 1
}

arg → tipo_arg id {
Si no STS.getTop().existe(id) Entonces
    STS.getTop().append(id, tipo.type, dir, 'arg', NULO, NULO)
    dir = dir + STT.getTop().getTam(tipo.type)
    arg.type = tipo.type
Sino
    error("El identificador ya fue declarado")
Fin Si
}

tipo_arg → base param_arr {
baseGBL = base.base
tipo_arg.type = param_arr.type
}

param_arr → [ ] $param\_arr_1$ {param_arr.type = STT.getTop().insert('array), 0, param_arr1.tipo)}

param_arr → {param arr.type = baseGBL}

sentencias → $sentencias_1$ sentencia {
L = newLabel()
backpatch(sentencias1.nextlist, L)
genCode(label L)
}

sentencia → si e_bool entonces $sentencia_1$ fin {
L = newLabel()
backpatch(e bool.truelist, L)
sentencia.nextlist =combinar(e bool.falselist, Sentencia1.nextlist)
genCode(label L)
}

sentencia → si e_bool entonces $sentencia_1$ sino $sentencia_2$ fin {
L1 = newLabel()
L2 = newLabel()
backpatch(e bool.truelist, L1)
backpatch(e bool.falselist, L2)
sentencia.nextlist = combinar(sentencia1.nextlist, sentencia2.nextlist)
genCode(label L1)

genCode(’goto’ sentencia1.nextlist[0])
genCode(label L2)
}

sentencia → mientras e_bool hacer $sentencia_1$ fin {
L1 = newLabel()
L2 = newLabel()
backpatch(sentencia1.nextlist, L1)
backpatch(e bool.truelist, L2)
sentencia.nextlist = e bool.falselist
genCode(label L1)
genCode(label L2)
genCode(’goto’ sentencia1.nextlist[0])
}

sentencia → hacer $sentencia_1$ mientras e_bool; {
L = newLabel()
genCode(”label” L)
batckbatch(sentencia1.nextlist, L)
}

sentencia → segun (variable) hacer casos predeterminado fin {
prueba = combinar(casos.prueba,predeterminado.prueba)
backpatch(casos.nextlist, L2)
sustituir(”??”, variable.dir, prueba)
}

sentencia → variable := expresion ; {
dir = reducir(expresion.dir, epresion.type, variable.type)
Si variable.code est = true Entonces
  genCode(variable.base’[’variable.des’]’ ’=’ dir)
Sino
  genCode(variable.dir ’=’ dir)
Fin Si
}

sentencia → escribir expresion ; {gen(”write” expresion.dir)}

sentencia → leer variable ; {gen(”read” variable.dir)}

sentencia → devolver ; {genCode(”return”)}

sentencia → devolver expresion ; {
index = newIndex()
sentencia.nextlist = newIndexList(index)
genCode(”goto” index)
}

sentencia → terminar ; {
index = newIndex()
sentencia.nextlist = newIndexList(index)
genCode(”goto” index)
}

sentencia → inicio sentencias fin {sentencia.nextlist = sentencia1.nextlist}

casos → caso num : sentencia casos1 {
casos.nextlist =
combinar(casos.nextlist, sentencia1.nextlist)
L = newLabel()
/*Indica el inicio del c´odigo para la sentencia*/
genCode(”label” L)
casos.prueba = casos1.prueba
casos.prueba.append(if ”??” ”==” num.dir ”goto” L)
}

casos → caso num : sentencia {
casos.prueba = newCode()
L = newLabel()
/*Indica el inicio del c´odigo para la sentencia*/
genCode(”label” L)
casos.prueba.append(if ”??” ”==” num.dir ”goto” L)
casos.nextlist = sentencia.nextlist
}

predeterminado → pred : sentencia {
predeterminado.prueba = newCode()
L = newLabel()
/*Indica el inicio del c´odigo para la sentencia*/
genCode(”label” L)
predeterminado.prueba.append(”goto” L)
}

predeterminado → {pretederminado.prueba = NULO}

e_bool → e_bool_1 o e_bool_2 {
L = newLabel()
backpatch(e_bool1.falselist, L)
e_bool.truelist = combinar(e_bool1.truelist,e_bool2.truelist)
e_bool.falselist = e_bool2.falselist
genCode(label L)
}

e_bool → e_bool_1 y e_bool_2 {
L = newLabel()
backpatch(e_bool1.truelist, L)
e_bool.truelist = e_bool1.truelist
e_bool.falselist = combinar(e_bool1.falselist,e_bool2.falselist)
genCode(label L)
}

e_bool →no e_bool_1 {
e_bool.truelist = e_bool1.falselist
e_bool.falselist = e_bool.truelist
}

e_bool → relacional {
e_bool.truelist = relacional op.truelist
e_bool.falselist = relacional op.falselist
}

e_bool → verdadero {
index0 = newIndex()
e_bool.truelist = newIndexList(index0)
genCode(’goto’ index0)
}

e_bool → falso {
index0 = newIndex()
e_bool.falselist = newIndexList(index0)
genCode(’goto’ index0)
}

relacional → relacional1 > relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode(’if’ relacional1.dir > relacional2 ’goto’ index0)
genCode(’goto’ index1)
}

relacional → relacional1 < relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode(’if’ relacional1.dir < relacional2 ’goto’ index0)
genCode(’goto’ index1)
}

relacional → relacional1 <= relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode(’if’ relacional1.dir <= relacional2 ’goto’ index0)
genCode(’goto’ index1)
}

relacional → relacional1 >= relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode(’if’ relacional1.dir >= relacional2 ’goto’ index0)
genCode(’goto’ index1)
}

relacional → relacional1 < > relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode(’if’ relacional1.dir <> relacional2 ’goto’ index0)

```
genCode('goto' index1)
}
```

```
relacional → relacional1 = relacional2 {
index0 = newIndex()
index1 = newIndex()
relacional.truelist = newIndexList(index0)
relacional.falselist = newIndexList(index1)
genCode('if' relacional1.dir = relacional2 'goto' index0)
genCode('goto' index1)
}
```

```
relacional →  expresion {
relacional.dir = expresion.dir
relacional.tipo = expresion.tipo
}
```

```
expresion →  expresion1 + expresion2 {
expresion.type = max(expresion1.type, expresion2.type)
expresion.dir = newTemp()
dir1 = ampliar(expresion1.dir, epxresion1.type,expresion.type)
dir2 = ampliar(expresion2.dir, epxresion2.type,expresion.type)
getCode(expresion.dir '=' dir1 '+' dir2)
}
```

```
expresion →  expresion1 - expresion2 {
expresion.type = max(expresion1.type, expresion2.type)
expresion.dir = newTemp()
dir1 = ampliar(expresion1.dir, epxresion1.type,expresion.type)
dir2 = ampliar(expresion2.dir, epxresion2.type,expresion.type)
getCode(expresion.dir '=' dir1 '-' dir2)
}
```

```
expresion →  expresion1 * expresion2 {
expresion.type = max(expresion1.type, expresion2.type)
expresion.dir = newTemp()
dir1 = ampliar(expresion1.dir, epxresion1.type,expresion.type)
dir2 = ampliar(expresion2.dir, epxresion2.type,expresion.type)
getCode(expresion.dir '=' dir1 '*' dir2)
}
```

```
expresion →  expresion1 / expresion2 {
expresion.type = max(expresion1.type, expresion2.type)
expresion.dir = newTemp()
dir1 = ampliar(expresion1.dir, epxresion1.type,expresion.type)
dir2 = ampliar(expresion2.dir, epxresion2.type,expresion.type)
getCode(expresion.dir '=' dir1 '/' dir2)
}
```

```
expresion →  expresion1 % expresion2 {
Si expresion1.type = entero and expresion2.type = entero Entonces
   expresion.type = max(expresion1.type,expresion2.type)
   expresion.dir = newTemp()
   dir1 = ampliar(expresion1.dir, epxresion1.type,expresion.type)
```

```
   dir2 = ampliar(expresion2.dir, epxresion2.type,expresion.type)
   getCode(expresion.dir '=' dir1 '+' dir2)
Sino
  error("No se puede obtener el modulo si los operandos no son enteros")
Fin Si
}
```

```
expresion → ( expresion1 ) {
expresion.type = expresion1.type
expresion.dir = expresion1.dir
}
```

```
expresion → variable {
expresion.type = variable.type
expresion.dir = variable.dir
}
```

```
expresion → num {
expresion.type = num.type
expresion.dir = num.dir
}
```

```
expresion → cadena {
expresion.type ='string'
Si TablaCadenas.existe(cadena) Entonces
    expresion.dir= TablaCadena.getIndexStr(cadena)
Sino
    expresion.dir=TablaCadena.insert(cadena)
Fin Si
}
```

```
expresion → caracter {
expresion.type ='car'
Si TablaCadenas.existe(car) Entonces
    expresion.dir= TablaCadena.getIndexStr(car)
Sino
    expresion.dir=TalbaCadena.insert(car)
Fin Si
}
```

```
variable →  id variable_comp {
Si STS.getTop().existe(id) Entonces
    IDGBL = id
    Si variable_comp.code_est = true Entonces
        variable.dir=newTemp()
        variable.type = variable_comp.type
        genCode(variable.dir '=' id'[' variable_comp.des']')
        variable.base = id.dir
        variable.code_est= true
        variable.des = variable_comp.des
Sino
       variable.dir = id)
       variable.type = STS.getTop().getType(id)
       variable.code_est= false
```

```
Sino
      error(“No existe la variable”)
Fin Si
}
```

```
variable_comp →  dato_est_sim {
variable_comp.type = dato_est_sim.type
variable_comp.des = dato_est_sim.des
variable_comp.code_est = dato_est_sim.code_est
}
```

```
variable → arreglo {
variable_comp.type = arreglo.type
variable_comp.des = arreglo.dir
variable_comp.code_est = true
}
```

```
variable → ( parametros ) {
Si STS.getGlobal().getVar(IDGBL)= ’func’ Entonces
    lista = STS.getGlobal().getListaArgs(IDGBL)
    num = STS.getGlobal().getNumArgs(ID)
    Si num = parametros.num Entonces
       Para cada i = 0 hasta i = num hacer
          Si lista[i] !=parametros.lista[i] entonces
             Error(“Los parámetros pasados no coinciden con los parámetros de la función”)
Fin si
}
```

```
dato_est_sim → dato_est_sim.id {
Si dato_est_sim1.estructura = true Entonces
   Si dato_est_sim1.tabla.existe(id) Entonces
      dato_est_sim.des = dato_est_sim1.des + dato_est_sim.tabla1.getDir(id)
      typeTemp=dato_est_sim1.tabla.getType(id)
      estTemp = dato_est_sim1.tabla.tablaTipos.getName(typeTemp)
      Si estTemp = ’struct’ Entonces
         dato_est_sim.estructura= true
         dato_est_sim.tabla= dato_est_sim.tabla.tablaTipos.getTipoBase(typeTemp).tabla
      Sino
         dato_est_sim.estructura= false
         dato_est_sim.tabla= NULO
         dato_est_sim.type = dato_est_sim1.tabla.getType(id)
      FinSi
   dato est sim.code est=true
   Sino
        error(“No existe estructura con ese id”)
   FinSi
Sino
   error(“No existe la estructura”)
FinSi
}
```

```
dato_est_sim →  ε {
typeTemp = STS.getTop().getType(id)
Si STT.getTop().getName(typeTemp) =’struct’ Entonces
```

```
    dato_est_sim.estructura= true
    dato_est_sim.tabla= STT.getTop().getTipoBase(typeTemp).tabla
    dato_est_sim.des = 0
Sino
    dato_est_sim.estructura= false
    dato_est_sim.type = STT.getTop().getType(id)
Fin Si
dato_est_sim.code est=false
}
```

```
arreglo → [expresion] {
arreglo.type = STS.getTop().getType(IDGBL)
Si STT.getTop().getName(arreglo.type) = 'array' Entonces
   Si expresion.type = entero Entonces
      typeTemp = STT.getTop().getTypeBase(arreglo.type)
      tam = STT.getTop().getTam(typeTemp)
      arreglo.dir = newTemp()
      genCode(arreglo.dir'=' expresion.dir '*' tam)
   Sino
      error("La expresión no es de tipo entero")
   Fin Si
Sino
   error("No existe el arreglo")
Fin Si
}
```

```
arreglo → arreglo1 [expresion] {
arreglo.type = STS.getTop().getType(arreglo1.type)
Si STT.getTop().getName(arreglo.type) = 'array' Entonces
   Si expresion.type = entero Entonces
      typeTemp = STT.getTop().getTypeBase(arreglo.type)
      tam = STT.getTop().getTam(typeTemp)
      dirTemp = newTemp()
      arreglo.dir = newTemp()
      genCode(dirTemp'=' expresion.dir '*' tam)
      genCode(arreglo.dir'=' arreglo1.dir '+' dirTemp)
   Sino
      error("La expresión no es de tipo entero")
   Fin Si
Sino
  error("No existe el arreglo")
Fin Si
}
```

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

```
parametros → ε {
parametros.lista = NULO
parametros.num = 0
}
```

lista_param → *lista_param* $_1$, expresion {
lista_param.lista = lista_param1.lista
lista_param.lista.append(expresion.type)
lista_param.num = lista_param1 + 1
}

lista_param → expresion {
lista_param.lista = newList()
lista_param.lista.append(expresion.type)
lista_param.num = 1
}