Universidad Nacional Autónoma de México
Facultad de Ciencias
Compiladores
Proyecto Final

Jesús Fernando Moreno Ruíz
414001967

13/12/19

## 1 Esquema de Traducción

| |
|---|
| programa → declaraciones { dir = 0 } funciones { StackTT = newStackTT()<br>StackTS = newStackTS()<br>ts = newSymTab()<br>tt = newTypeTab()<br>StackTT.push(tt)<br>StackTS.push(ts)<br>TablaDeCadenas = newTablaCadenas() } |
| declaraciones → tipo lista_var \n declaraciones { type = tipo.tipo } |
| declaraciones → tipo_registro lista_var \n declaraciones { type = tipo_registro.tipo } |
| declaraciones → $\epsilon$ { Hacer nada } |
| tipo_registro → registro \n inicio *declaraciones*\n fin { ts = newSymTab()<br>tt = newTypeTab()<br>StackDir.p0ush(dir)<br>dir = 0<br>StackTT.push(tt)<br>StackTS.push(ts)<br>dir = StackDir.pop()<br>tt1 = StackTT.pop()<br>StackTS.getCima().setTT(tt1)<br>ts1 = StackTS.pop()<br>dir = StackDir.pop()<br>type = StackTT.getCima().addTipo("registro",0,ts1) } |
| tipo → base tipo_arreglo { base = base.tipo<br>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] $tipo\_arreglo_1$ { si $num.tipo = ent$ y $num.val > 0$ entonces<br>\| tipo_arreglo.tipo = StackTT.getCima().addTipo("array", num.val,<br>\| $tipo\_arreglo_1$.tipo )<br>en otro caso<br>\| Error("El indice tiene que ser entero y mayor que cero")<br>fin} |
| tipo_arreglo → $\epsilon$ { tipo_arreglo.tipo = base } |
| lista_var → $lista_var$, id { si $StackTS.getCima().getId(id.lexval) = -1$ entonces<br>\| StackTS.getCima().addSym(id.lexval, tipo, dir, "var")<br>\| dir = dir + StackTT.getCima().getTam(tipo)<br>en otro caso<br>\| Error("El identificador ya fue declarado")<br>fin} |

```
lista_var → id { si StackTS.getCima().getId(id.lexval) = -1 entonces
                | StackTS.getCima().addSym(id.lexval, tipo, dir, "var")
                | dir = dir + StackTT.getCima().getTam(tipo)
                en otro caso
                | Error("El identificador ya fue declarado")
                fin}
```

```
funciones → func tipoid  { si StackTS.getFondo().getId(id.lexval) ≠ -1 entonces
                         | StackTS.getFondo().addSym(id.lexval, tipo, --, "func")
                         | StackDir.push(dir)
                         | FuncType = tipo.tipo
                         | FuncReturn = false
                         | dir = 0
                         | StackTT.push(tt)
                         | StackTS.push(ts)
                         | dir = StackDir.pop()
                         | add_quad(code,'label',-,-,id.lexval)
                         | L = newLabel()
                         | backpatch(code,sentencias.next,L)
                         | add_quad(code,'label',-,-,L)
                         | StackTT.pop()
                         | StackTS.pop()
                         | dir = StackDir.pop()
                         | StackTS.getCima().addArgs(id.lexval,argumentos.lista)
                         | si (tipo.tipo ≠ sin) y (FuncReturn = false) entonces
                         |   Error("la funcion no tiene valor de retorno")
                         | endif
                         en otro caso
                         | Error("El identificador ya fue declarado")
                         fin }

            ( argumentos )  inicio  \n declaraciones sentencias \n  fin  \n funciones
```

```
funciones → ϵ { Hacer Nada }
```

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

```
argumentos → sin { argumentos.lista = nulo }
```

```
lista_arg → lista_arg1 arg { lista_arg.lista = lista_arg1.lista
                             lista_arg.lista.add(arg.tipo) }
```

```
lista_arg → arg { lista_arg.lista = newListaParam()
                  lista_arg.lista-add(arg.tipo) }
```

```
arg → tipo_arg id { si StackTS.getCima().getId(id.lexval) = -1 entonces
                    | StackTS.getCima().addSym(id.lexval, tipo, dir, "var")
                    | dir = dir + StackTT.getCima()-getTam(tipo)
                    en otro caso
                    | Error("El identificador ya fue declarado")
                    fin
                   arg.tipo = tipo_arg.tipo }
```

```
tipo_arg → base param_arr { base = base.tipo
                            tipo_arg.tipo = param_arr.tipo }
```

```
param_arr → [] param_arr1 { param_arr.tipo = StackTT.getCima().addTipo("array", -, param_arr1.tipo) }
```

```
param_arr → ϵ { param_arr.tipo = base }
```

```
sentencias → sentencias \n sentencia
                        { L = newLabel()
                        backpatch(code, sentencias.listnext, L)
                        sentencias.listnext = combinar(expresion_booleaba.listfalse, sentencias.listnext) }
```

```
sentencias → sentencia { sentencias.listnext = sentencia.listnext }
```

```
sentencia → si esprexion_booleana {L = newLabel()
                                   backpatch(code, expresion_booleana.listtrue, L)} \n
               sentencias
               { sentencia.listnext = combinar(expresion_booleana.listfalse, sentencias.listnext) }\n fin
```

```
sentencia → si expresion_booleana { L = newLabel()
                                    L1 = newLabel()
                                    backpatch(code, expresion_booleana.listtrue, L)
                                    backpatch(code, expresion_booleana.listfalse, L1) } \n
            sentencias1 \n sino \n sentencias2
            {sentencia.listnext = combinar(sentencias1.listnext, sentencias2.listnext )} \n fin
```

```
sentencia → mientras \n expresion_booleana { L = newLabel()
                                             L1 = newLabel()} hacer \n sentencias
                                             { backpatch(code, sentencias.listnext, L)
                                             backpatch(code, expresion boolean.listtrue, L1)
                                             sentencia.listnext = expresion booleana.listfalse
                                             add quad(code, "goto", -, -, L) } \n fin
```

```
sentencia → hacer \n sentencias_2 \n
            mientras que expresion_booleana { L = newLabel()
                                              backpatch(code, expresion_boolean.listtrue, L)
                                              backpatch(code, sentencias.listnext, L1)
                                              sentencia.listnext = expresion_booleana.listfalse
                                              add_quad(code, "label", -, -, L) }
```

```
sentencia → id := expresion { si StackTS.getCima().getId(id.lexval) ≠ -1 entonces
                            | t =   StackTS.getCima().getTipo(id.lexval)
                            | d = StackTS.getCima().getDir(id.lexval)
                            | α = reducir(expresion.dir, expresion.tipo. variable.tipo)
                            | add_quad(code, "=", α, -, "id" + d)
                            en otro caso
                            | Error("El identificador no ha sido declarado")
                            fin
                            sentencia.listnext = nulo }
```

```
sentencia → variable := expresion { α = expresion.dir, expresion.tipo, variable.tipo)
                                    add_quad(code, " = ", α, -, variable.base[variable.dir])
                                    sentencia.listnext = nulo}
```

```
sentencia → escribir expresion { add_quad(code, "print", expresion.dir, -,-)
                                 sentencia.listnext = nulo}
```

```
sentencia → leer variable { add_quad(code, "print", variable.dir, -,-)
                            sentencia.listnext = nulo }
```

```
sentencia → devolver { si FuncType = sin entonces
                     | add_quad(code, "return", -, -, -)
                     en otro caso
                     | 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
                     en otro caso
                     | 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) }
```

```
expresion_booleana → expresion_booleana_1 yy expresion_booleana_2
                   { L = newLable()
                   backpatch(code, expresion_booleana_1.listtrue, L)
                   expresion_booleana.listtrue = expresion_booleana_2.listtrue
                   expresion_booleana.listfalse =
                   combinar(expresion_booleana_1.listfalse,expresion_booleana_2.listfalse
                   add_quad(code, "label", -, -, L) }
```

```
expresion_booleana → no expresion_booleana_1 { expresion_booleana.listtrue = expresion_booleana_1.listfalse
                                               expresion_booleana.listfalse = expresion_booleana_1.listtrue}
```

```
expresion_booleana → verdadero { I = newIndex()
                                 expresion_booleana.listtrue = newList()
                                 expresion_booleana.listtrue.add(I)
                                 add_quad(code, "goto",-,-,I)
                                 expresion_booleana.listfalse = nulo }
```

```
expresion_booleana → falso { I = newIndex()
                             expresion_booleana.listtrue = nulo
                             expresion_booleana.listfalse = newList()
                             expresion_booleana.listfalse.add(I)
                             add_quad(code, "goto",-,-,I) }
```

```
relacional → relacional1 < relacional2 { 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 → relacional1 > relacional2 { 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 → relacional1 <= relacional2 { 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 → relacional1 >= relacional2 { 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 → relacional1 == relacional2 { 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 → relacional1 <> relacional2 { 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 → expresion1 + expresion2 { 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 → expresion1 - expresion2 { 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 → expresion1 * expresion2 { 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 → expresion1 / expresion2 { 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 → expresion1 % expresion2 { 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 → (expresion1) { expresion.dir = expresion1.dir
                           expresion.tipo = expresion1.tipo }
expresion → variable { expresion.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}
hline expresion → cadena { expresion.tipo = cadena
                           expresion.dir = TablaDeCandenas.add(cadena)}
expresion → num { expresion.tipo = caracter
                  expresion.dir = TablaDeCadenas.add(caracter)}
```

```
expresion → id(parametros) { si StackTS.getFondo().getId(id.lexval) ≠ -1 entonces
                           | si StackTS.getFondo().getVar(id.lexval) = "func" entonces
                           | | lista = StackTs.getFondo().getArgs(id.lexval)
                           | | si lista.getTam() ≠ parametros.getTam() entonces
                           | | | Error(El numero de argumentos no coincide)
                           | | fin
                           | | para i = 0, i < parametros.lista.getTam(), 1 hacer
                           | | | si parametros[i] ≠ lista[i] entonces
                           | | | | Error("El tipo de los parametros no coinciden")
                           | | | fin
                           | | fin
                           | | expresion.dir = newTemp()
                           | | expresion.tipo = StackTs.getFondo().getTipo(id.lexval)
                           | | add_quad(code, "=", "call", id.lexval, expresion.dir)
                           | en otro caso
                           | | Error("El identificador no ha sido declarado")
                           fin }
```

```
variable → arreglo { variable.dir = arreglo.dir
                     variable.base = arreglo.base
                     variable.tipo = arreglo.tipo }
```

```
variable → id1.id2 { si StackTS.getFondo().getId(id.lexval) ≠ -1 entonces
                   | t = StackTS.getFondo().getTipo(id.lexval)
                   | t1 = StackTT.getFondo().getTipo(t)
                   | si t1 = "registro" entonces
                   | | tipoBase = StackTT.getFondo().getTipoBase(t)
                   | | si tiposBase.getId(id2) ≠ -1 entonces
                   | | | variable.tipo = tipoBase.getType(id2
                   | | | variable.dir = id2
                   | | | variable.base = id1
                   | | en otro caso
                   | | | Error("El id no existe en la estructura")
                   | | fin
                   | en otro caso
                   | | Error("El id no es una estructura")
                   | fin
                   En otro caso
                   | Error("El identificador no ha sido declarado")
                   fin }
```

```
arreglo → id[expresion] { si StackTS.getCima().getId(id.lexval) ≠ -1 entonces
                        | t = StackTS.getCima().getTipo(id.lexval)
                        | si StackTs.getCima().getTipo(t) == "array" entonces
                        | | si expresion.tipo = ent entonces
                        | | | arreglo.base = id.lexval
                        | | | arreglo.tipo = StackTT.getCima().getTipoBase(t)
                        | | | arreglo.tam = StackTT.getCima().getTipoi(arreglo.tipo)
                        | | | arreglo.dir = newTemo()
                        | | | add_quad(code, "*", expresion.dir, arreglo.tam, arreglo.dir)
                        | | en otro caso
                        | | | Error (La expresion para un indice debe ser de tipo entero)
                        | | fin
                        | en otro caso
                        | | Error(El identificador no es un arreglo)
                        | fin
                        fin
                        en otro caso
                        | Error("El identificador no ha sido declarado")
                        fin }
```

```
arreglo → arreglo1 [expresion] { si StackTT.getCima().getTipoBase(arreglo1.tipo) = "array" entonces
                                 | si expresion.tipo = ent entonces
                                 | | arreglo.base = arreglo1.base
                                 | | arreglo.tipo = StackTT.getCima().getTipoBase(arreglo1.tipo)
                                 | | arreglo.tam = StackTT.getCima().getTipo(arreglo.tipo)
                                 | | temp = newTemp()
                                 | | arreglo.dir = newTemp()
                                 | | add_quad(code, "*", expresion.dir, arreglo.tam, temp)
                                 | | add_quad(code, "+", arreglo1.dir, temp, arreglo.dir)
                                 | en otro caso
                                 | | Error (La expresion para un indice debe ser de tipo entero)
                                 | fin
                                 fin
                                 en otro caso
                                 | Error("El arreglo no tiene tantas dimensiones")
                                 fin }
parametros → lista_param { parametr5os.lista = lista_param.lista }
parametros → ϵ { parametros.lista = nulo }
lista_param → lista_param1 { lista_param.lista = newListaParam()
                             lista_param.lista.add(param.tipo)
                             add_quad(code, "param", expresion.dir, -, -) } , expresion
lista_param → expresion { lista_param.lista = newListaParam()
                          lista_param.lista.add(expresion.tipo)
```