UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

COMPILADORES

PROYECTO FINAL

ALUMNOS: FLORES GONZÁLEZ JESÚS EDUARDO

HERNANDEZ FONTES ALDO

RAMÍREZ VIRAMONTES JOSUÉ YAFTE

SANDOVAL LARA LESLY MAYTE

SEMESTRE: 2021-1

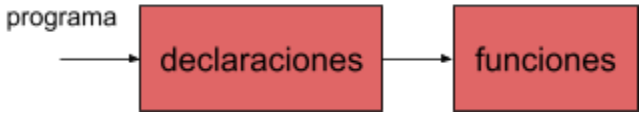
Indice

Gramática

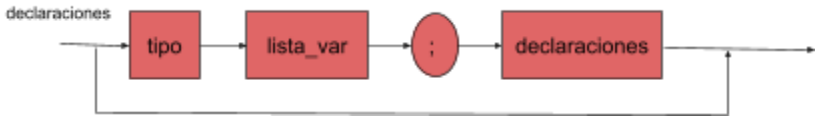
PRODUCCIÓN
programa → declaraciones funciones
declaraciones → tipo lista_var ; declaraciones | ε
tipo → basico compuesto
basico → **int** | **float** | **char** | **double**| **void**
compuesto → **(numero)** compuesto | ε
lista_var → lista_var , **id** | **id**
funciones → **func** tipo **id (** argumentos **)** bloque funciones | ε
argumentos→ lista_args | ε
lista_args→ lista_args, tipo **id** | tipo **id**
bloque → { declaraciones instrucciones }
instrucciones → instrucciones sentencia | sentencia
sentencia → parte_izquierda **=** bool ;| **if(** bool **)** sentencia
| **if(** bool **)** sentencia **else** sentencia | **while(** bool **)** sentencia
| **do** sentencia **while(** bool **)** | **break** ; | bloque | **return** exp ;| **return;**
| **switch(** bool **)** { casos }| **print** exp ; | **scan** parte_izquierda
casos → caso casos| ε | predeterminado
caso → **case numero:** instrucciones
predeterminado→ **default:** instrucciones
parte_izquierda → **id** localizacion |**id**
bool → bool || comb | comb
comb → comb **&&** igualdad | igualdad
igualdad → igualdad **==** rel | igualdad **!=** rel | rel
rel → exp $<$ exp | exp $<=$ exp | exp $>=$ exp |
| exp $>$ exp | exp
exp → exp + term | exp − term | term
term → term ∗ unario | term / unario | term **%** unario | unario
unario → **!**unario | − unario | factor
factor → **(**bool**)** | **id** localizacion | **numero** | **cadena** | **true**
| **false** | **id(**parametros**)**| **id**
parametros → lista_param | ε
lista_param → lista_param , bool | bool
localizacion→ localizacion **(** bool **)** | **(** bool **)**

Diagramas de sintaxis

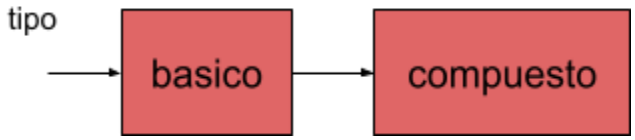
Programa → declaraciones funciones



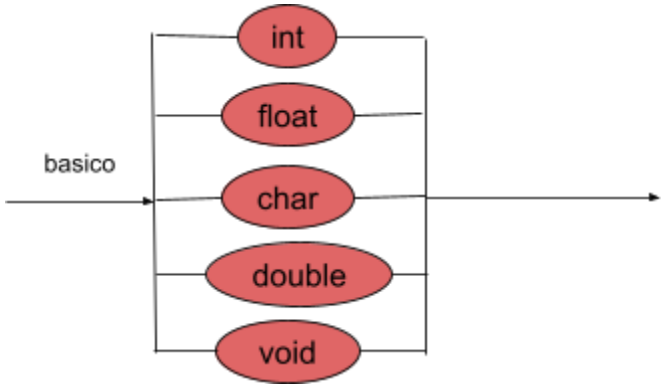
declaraciones→ tipo lista_var ; declaraciones| ε



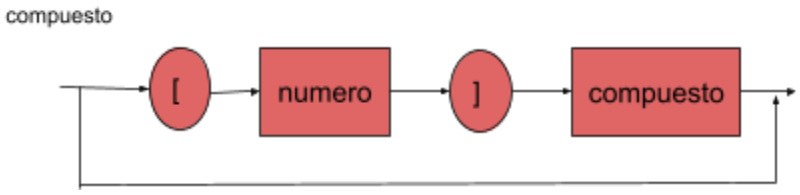
tipo → basico compuesto



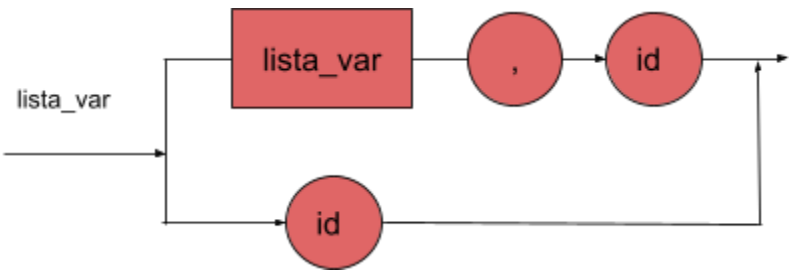
basico → int | float | char | double | void



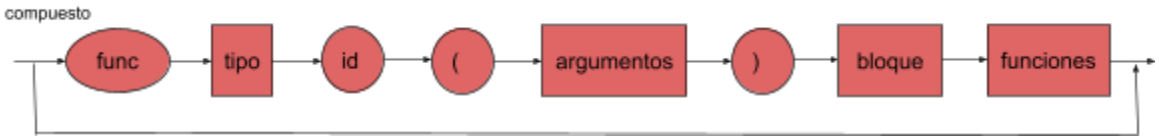
compuesto → [numero] compuesto | ε



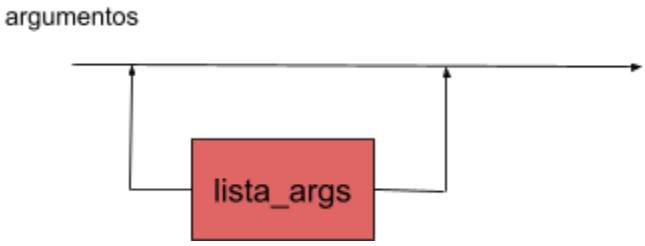
lista var → lista var , id | id



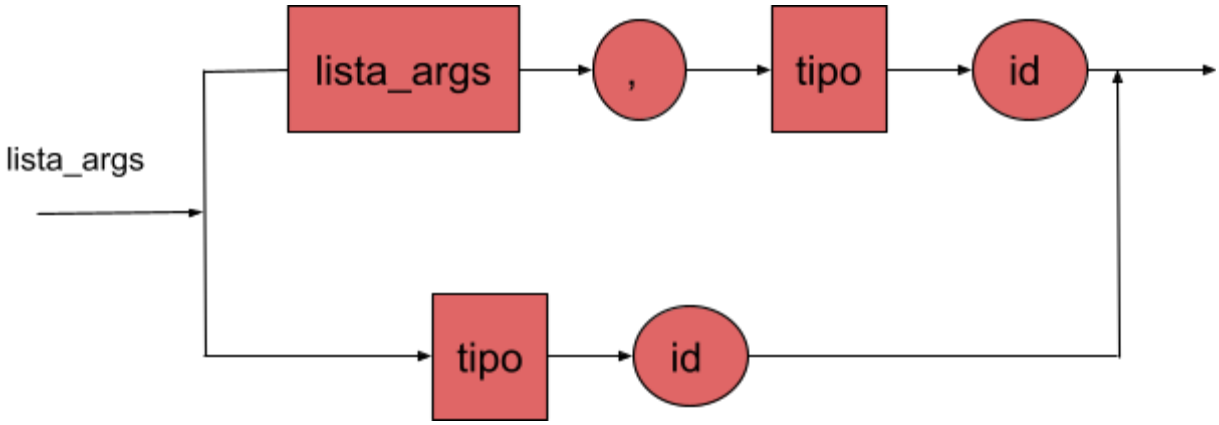
funciones → func tipo id (argumentos) bloque funciones|ε



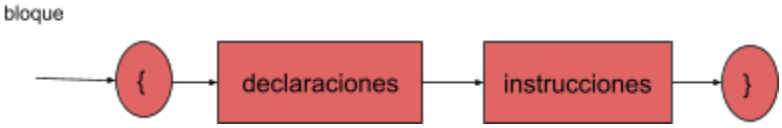
argumentos → lista_args| ε



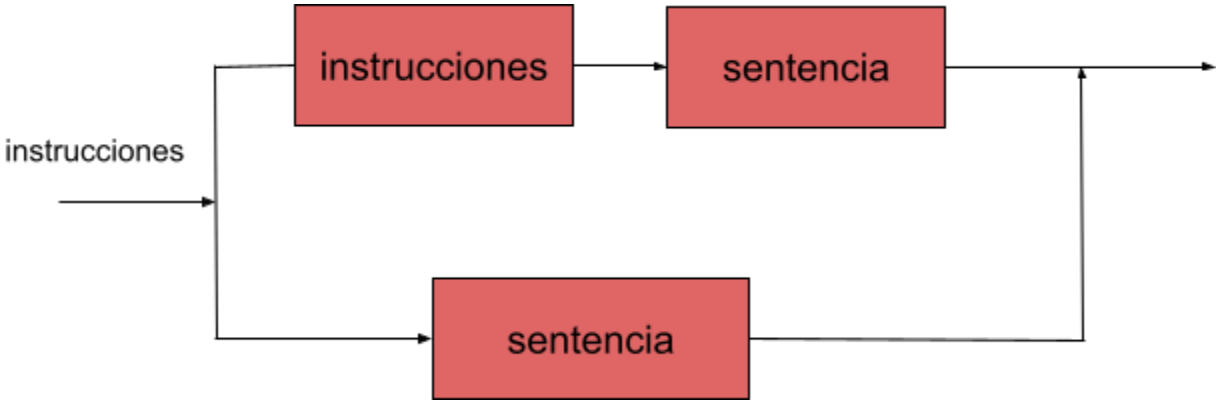
lista args → lista_args, tipo id | tipo id



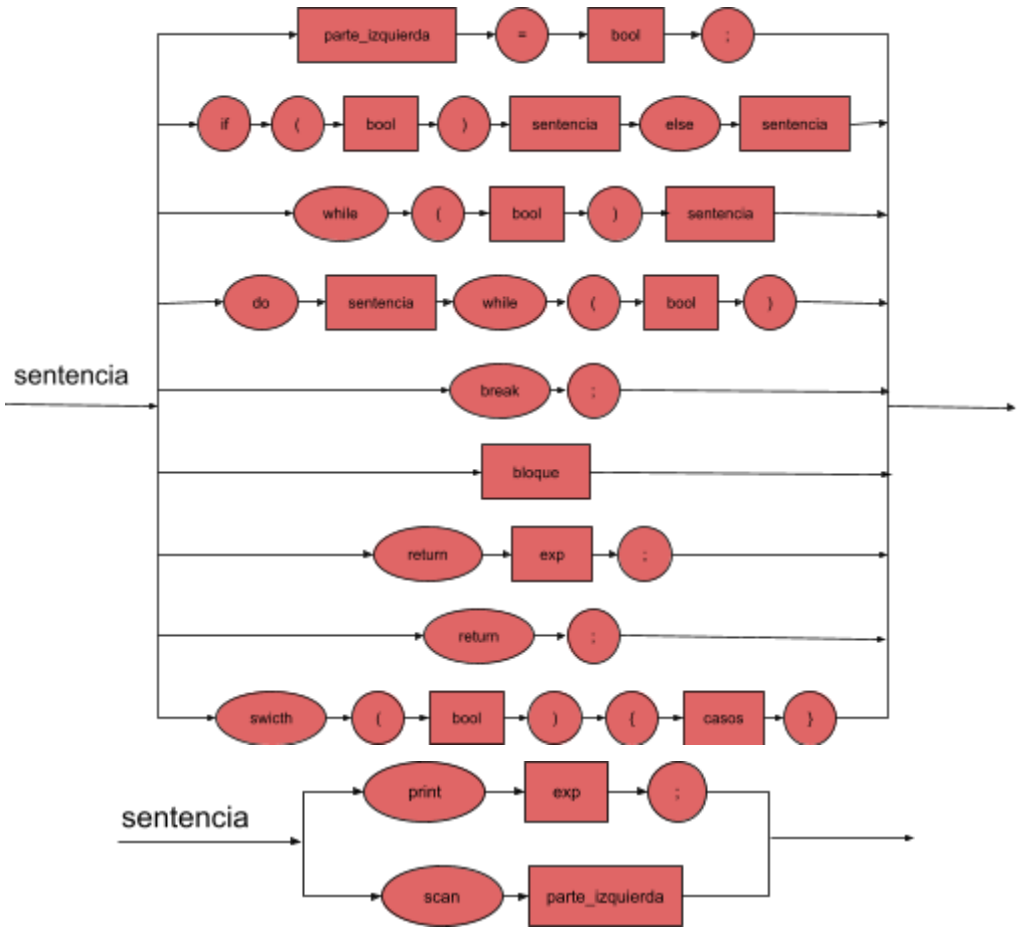
bloque → {declaraciones instrucciones}



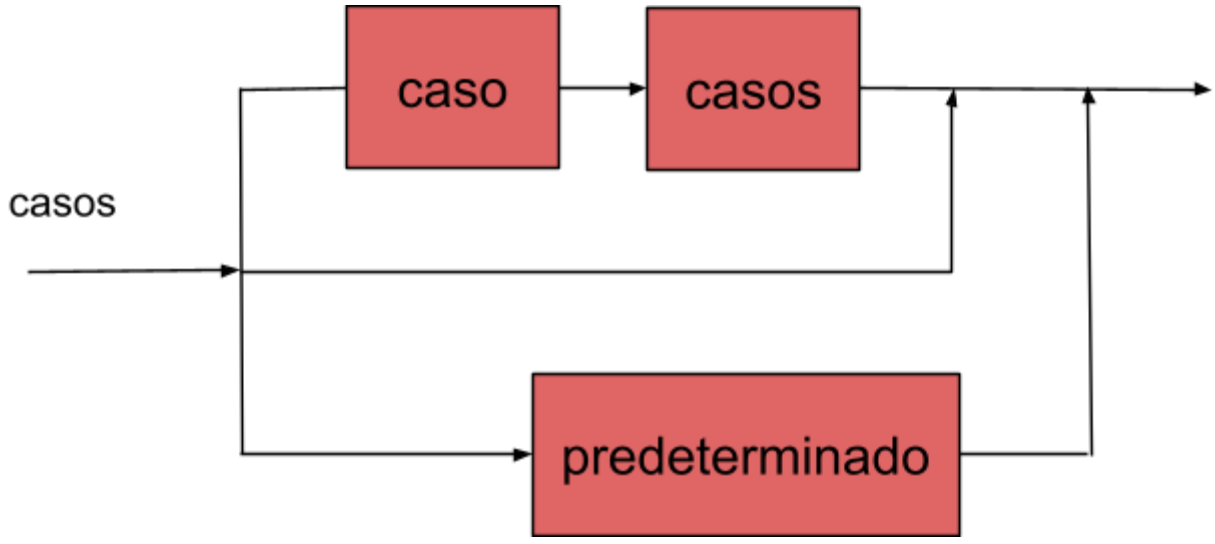
instrucciones → instrucciones sentencia | sentencia



sentencia → parte_izquierda = bool ;| if(bool) sentencia
| if(bool) sentencia else sentencia | while(bool) sentencia
| do sentencia while(bool) | break ; | bloque | return exp ;| return;
| switch(bool) {casos} | print exp;| scan parte_izquieda

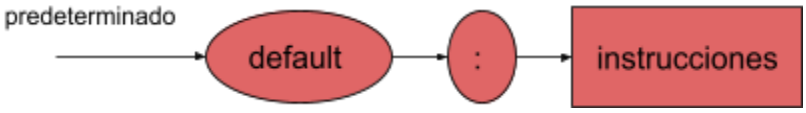


casos → caso casos| ε | predeterminado

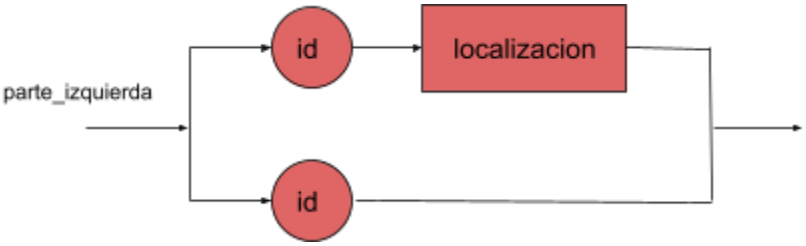


caso → case numero: instrucciones

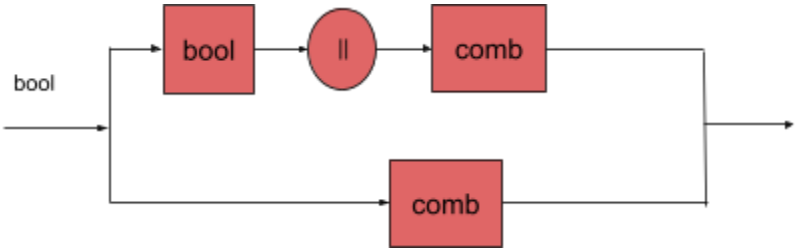
predeterminado → default: instrucciones



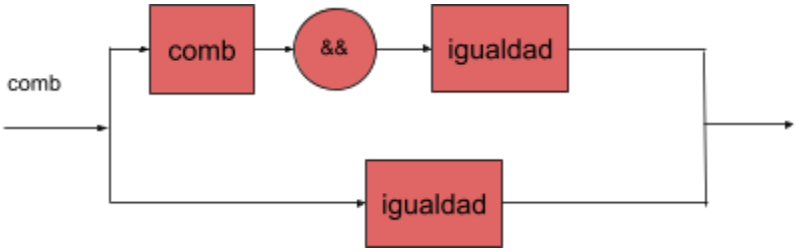
parte izquierda → id localizacion |id



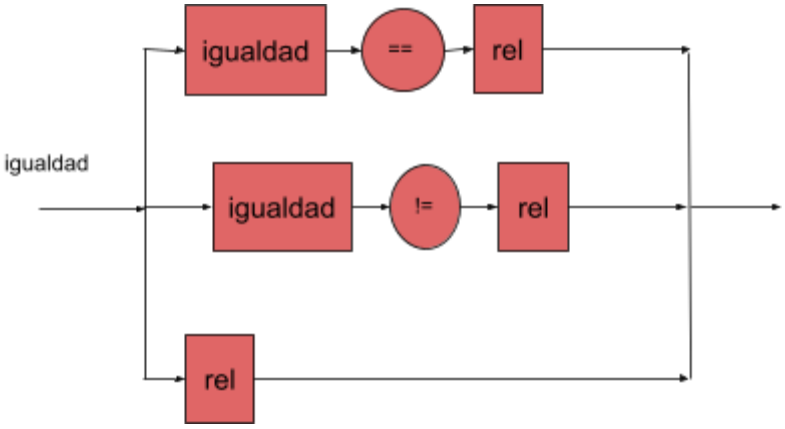
bool → bool || comb | comb



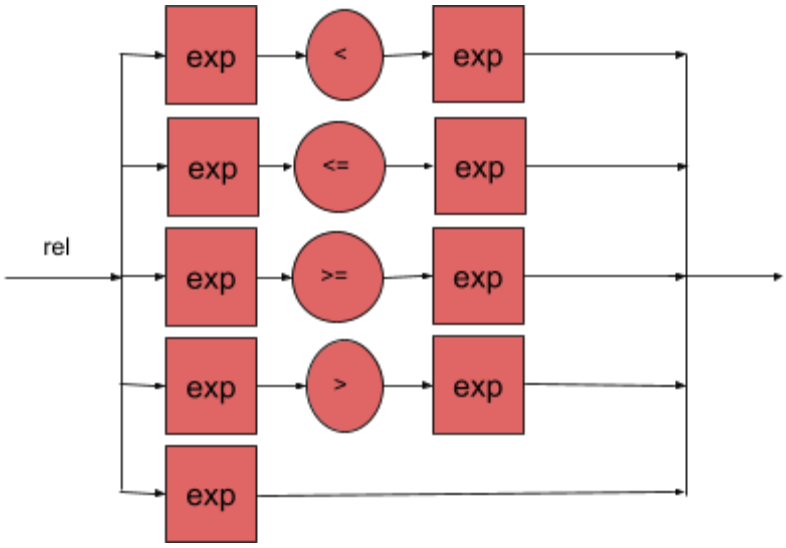
comb → comb && igualdad | igualdad



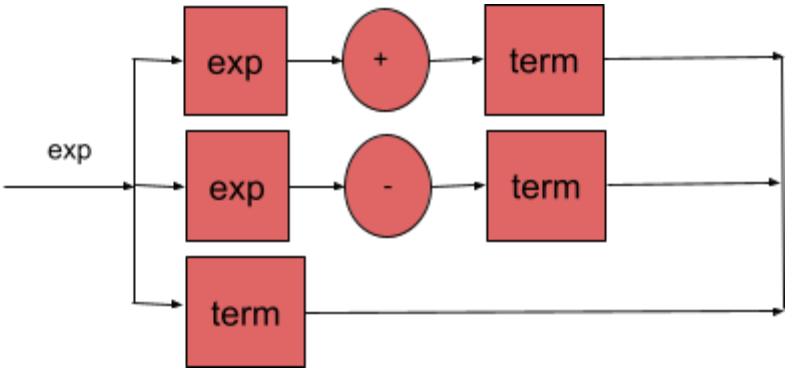
igualdad → igualdad == rel | igualdad != rel | rel



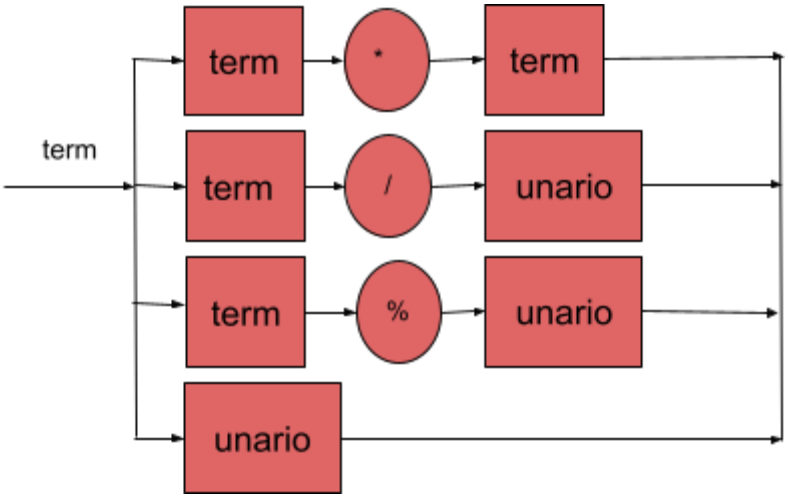
rel →exp < exp | exp <= exp | exp >= exp |
| exp > exp | exp



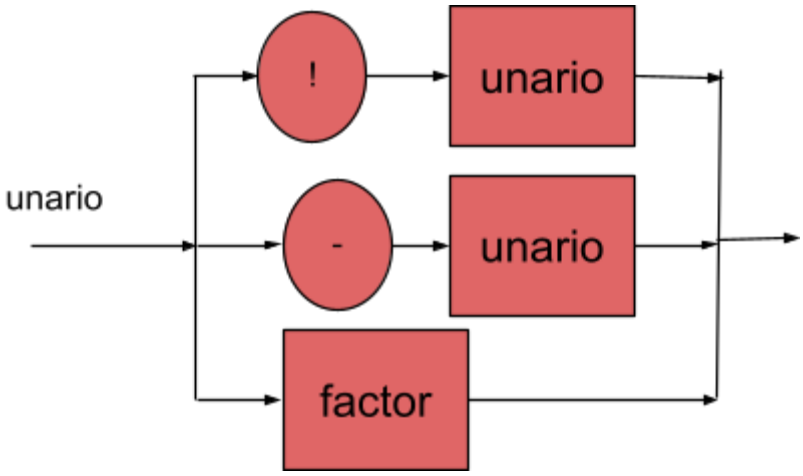
exp → exp + term | exp - term | term



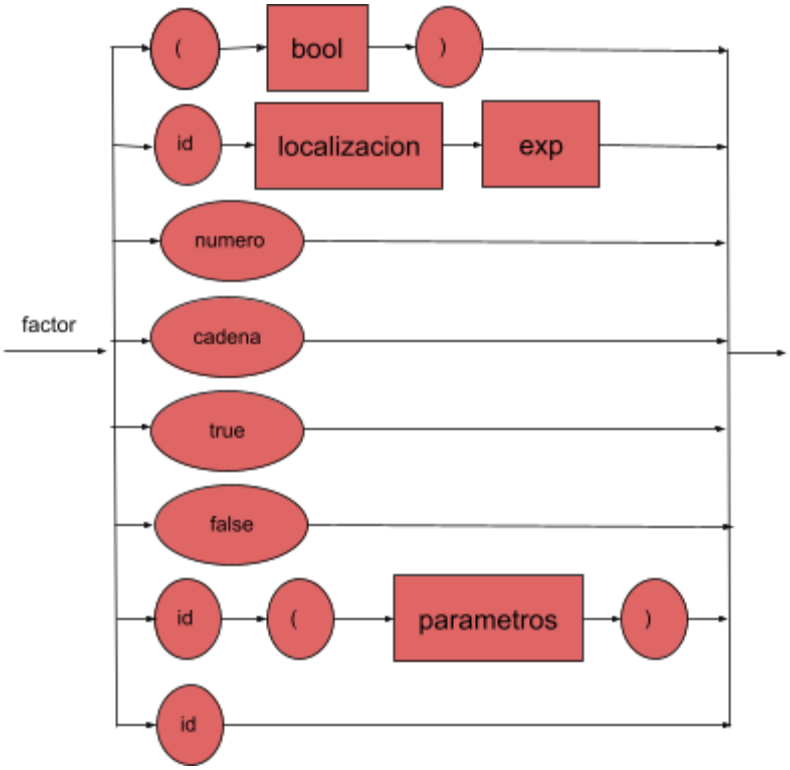
term → term * unario | term / unario| term % unario | unario



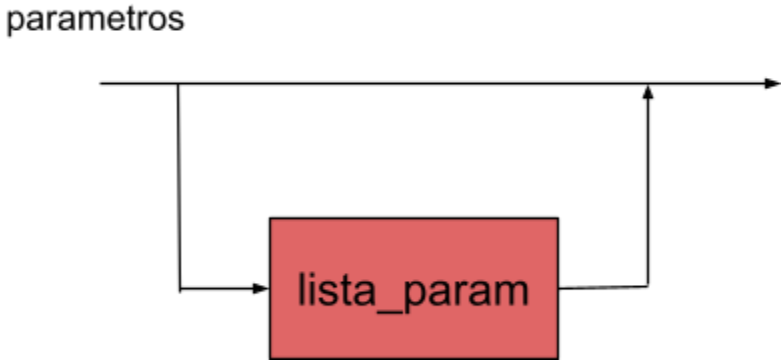
unario → !unario | - unario | factor



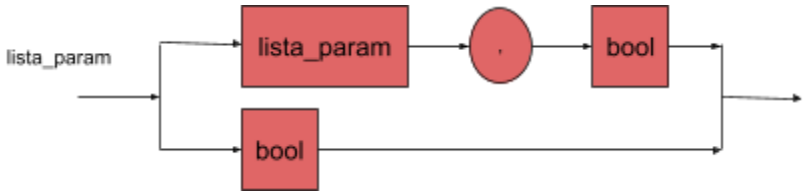
factor → (bool) | id localizacion | numero | cadena | true | false | id(parametros)| id



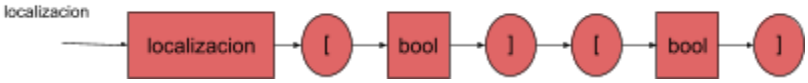
parametros → lista_param | ε



lista param → lista_param , bool | bool



localizacion → localizacion [bool] | [bool]



Diseño de las expresiones regulares

1. ## Identificadores

   Digito → [0-9]
   Letra →[a-zA-Z]
   Letras_ → ({Letra}|_)
   Identificador →{Letras_}({Digito}|{Letras_})*

2. ## Espacios

   espacio→[\r\n\tn]
   esp→[ ]
   esps → {espacio}+

3. ## Números enteros

   Enteros →((({Digito}+)(_)?({Digito}+))+)|({Digito})+

4. ## Números decimales

   Expo →[Ee][\+\-]?({Enteros})+
   Decimal→ (({Enteros}*\.{Digito}+)|({Enteros}+\.{Digito}*))
   ExpoFlo→(({Decimal})({Expo}?))|(({Enteros})({Expo}))

5. ## Cadenas

   Carac → [\\\"]
   Cadenas→ (\"({Letra}|{Digito}|{Carac}|{esp}|{OP})+\"|\'({Letra}|{Digito}|{Carac}|{espacio}|{OP})+\')

6. ## Booleanos

   True →"true"
   False → "false"

7. ## Operadores

   Asig → \=
   Parentesisabre→ \(
   Parentesiscierra → \)

Incremento → "++"
Disyuncion → "||"
And → "&&"
Igualque → "=="
Diferente → "!="
Menor→ "<"
Mayor→ ">"
Menorigual → "<="
Mayorigual → ">="
Suma →\+
Resta → \-
Mul → *
Div → \/
Modulo →\%
Negacion →\!
Direccionmem →\&

8. Palabras Reservadas

Int → int
Float → float
Char →char
Double→ double
While → while
Do →do
Switch→switch
Case→case
If→if
Else→else
Default→default
Break→break
Func→func
Void → void
Return → return
Print → print
Scan → scan

Eliminación de recursividad y factorización

Contienen recursividad y factorización:

- lista_var⟶ lista_var, id|id
    - recursividad
    - lista_var⟶ id lista_var'
    - lista_var'⟶, id lista_var'|ε

- lista_args⟶lista_args, tipo id| tipo id
    - recursividad
    - lista_args⟶tipo id lista_args'
    - lista_args'⟶, tipo id lista_args'| ε
- instrucciones ⟶ instrucciones sentencia|sentencia
    - recursividad
    - instrucciones ⟶sentencia instrucciones'
    - instrucciones' ⟶sentencia instrucciones'|ε
- parte_izquierda → **id** localizacion | **id**
    - Factorización
    - parte_izquierda→**id** parte_izquierda'
    - parte_izquierda'→localizacion | ε
- bool ⟶ bool || comb | comb
    - Recursividad
    - bool ⟶comb bool'
    - bool'⟶|| comb bool'|ε
- comb ⟶ comb && igualdad| igualdad
    - Recursividad
    - comb ⟶igualdad comb'
    - comb' ⟶&& igualdad comb'|ε
- igualdad⟶Igualdad==rel| igualdad!=rel|rel
    - Recursividad

    Igualdad ⟶rel igualdad'

    igualdad'⟶==rel igualdad' | != rel igualdad'| ε
- rel⟶exp<exp| exp<=exp| exp>=exp| exp>exp| exp
    - Factorización
    - rel⟶exp rel'
    - rel'⟶<exp|<=exp|>=exp|>exp| ε
- exp→exp+term|exp-term|term
    - Recursividad
    - exp → term exp'
    - exp' → +term exp' | -term exp' |ε
- term⟶term*unario|term/unario|term%unario|unario
    - Recursividad
    - term⟶unario term'
    - term' ⟶*unario term'| /unario term'|%unario term' |ε

- lista_param⟶lista_param,bool|bool
    - Recursividad izquierda
    - lista_param⟶bool lista_param’
    - lista_param’⟶,bool lista_param’|ε
- localizacion⟶ localizacion [bool] | [bool]
    - Recursividad izquierda
    - localizacion⟶[bool] localizacion’
    - localizacion’⟶[bool] localizacion’ | ε

Definición dirigida por sintaxis

Regla de producción	Reglas semánticas
programa⟶declaraciones funciones	PilaTS.push(nuevaTablaTS()) PilaTT.push(nuevaTablaTT()) dir = 0
declaraciones ⟶ tipo lista var ;declaraciones	lista var.tipo = tipo.tipo
declaraciones ⟶ ε	
tipo ⟶ basico compuesto	compuesto.base = basico.base tipo.tipo = compuesto.tipo
basico ⟶ int	basico.tipo = int
basico ⟶float	basico.tipo = float
basico ⟶ char	basico.tipo ⟶float
basico ⟶ double	base.tipo = double

basico → void	base.tipo = void
compuesto → [numero] compuesto1	compuesto.tipo = PilaTT.top().insertar(”array”, num.val, compuesto1.tipo) compuesto1.base = compuesto.base
compuesto → ε	compuesto.tipo = compuesto.base

lista_var→ id lista_var’	lista var’.tipo = lista var.tipo
	Si ! PilaTS.top().buscar(id) Entonces
	  PilaTS.top().insetar(id, lista var.tipo, dir, ”var”, NULO)
	  dir = dir + PilaTT.top().getTam(tipo.tipo)
	Sino
	  error(”El id no esta declarado”)
	FinSi

lista_var’→, id lista_var’	Si ! PilaTS.top().buscar(id) Entonces
	  PilaTS.top().insetar(id, lista var.tipo, dir, ”var”, NULO)
	  dir = dir + PilaTT.top().getTam(tipo.tipo)
	Sino
	  error(”El id no esta declarado”) ´
	FinSi

lista_var’→ε	lista_var’.tipo = lista_var’.base

funciones → func tipo id (argumentos) bloque funciones	ListaRetorno = NULO
	PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos)
	PilaDir.push(dir)
	dir = 0
	Si ! PilaTS.fondo().buscar(id) Entonces
	  Si equivalentesLista(ListaRetorno, tipo.tipo) Entonces
	  PilaTS.fondo().insetar(id, tipo.tipo, –, ’func’, argumentos.lista)
	  genCod(label(id))
	  bloque.sig = nuevaEtq()
	  genCod(label(bloque.sig))
	  Sino
	  error(”Los tipos de retorno no coinciden con el tipo de la funcion”)
	  FinSi
	Sino
	  error(”El id no esta declarado”)
	FinSi
	PilaTS.pop()
	PilaTT.pop()
	dir = PilaDir.pop()

funciones → ε	
argumentos→ lista args	argumentos.lista = lista args.lista
argumentos→ ε	**ar**gumentos.lista = NULO

lista_args→tipo id lista_args’	Si ! PilaTS.top().buscar(id) Entonces
	  PilaTS.top().insetar(id, tipo.tipo, dir, ”param”, NULO)
	  dir = dir + PilaTT.top().getTam(lista_var.tipo)
	Sino
	  error(”El id ya está declarado”)

	FinSi lista_args'.listaH = nuevaLista() lista_args'.listaH.agregar(tipo.tipo) Lista_args.lista = lista_args'.listaS
lista_args'→, tipo id lista_args'1	Si ! PilaTS.top().buscar(id) Entonces PilaTS.top().insetar(id, tipo.tipo, dir, "param", NULO) dir = dir + PilaTT.top().getTam(lista_var.tipo) Sino error("El id ya está declarado") FinSi lista_args´1.listaH = lista_args'.listaH lista_args'1.listaH.agregar(tipo.tipo) lista_args´.listaS = lista_args'1.listaS
lista_args'→ε	lista_args'.listaS = Lista_args'.listaH
bloque → { declaraciones instrucciones }	
instrucciones →sentencia instrucciones'	sentencia.sig = nuevaEtq() genCod(label(sentencia.sig))
instrucciones' →sentencia instrucciones'1	instrucciones'1.sig = nuevaEtq() genCod(label(instrucciones'1.sig))
instrucciones' →ε	
parte_izquierda→**id** parte_izquierda'	parte_izquierda'.base = id.lexval parte_izquierda.dir = parte_izquierda'.dir'.tipo parte_izqiuerda.tipo = parte_izquierda
parte_izquierda'→localizacion	localizacion.base =parte_izquierda'.base parte_izquierda'.dir = localizacion.dir parte_izqiuerda'.tipo = localizacion.tipo
parte_izquierda'→ε	Si PilaTS.top().buscar(parte_izquierda'.base) Entonces parte_izquierda'.dir = parte_izquierda'.base parte_izuierda'.tipo = PilaTS.top().getTipo(parte_izq'.dir) Sino error("El id no está declarado") FinSi

sentencia → parte_izquierda= bool ;	Si equivalentes(localizacion.tipo, bool.tipo) Entonces
	  d1 = reducir(bool.dir, bool.tipo, localizacion.tipo)
	  genCod(localizacion.dir '=' d1)
	Sino
	  error("Tipos incompatibles")
	FinSi

sentencia → if(bool) sentencia1	bool.vddr = nuevaEtq() bool.fls = sentencia.sig sentencia1.sig = sentencia.sig genCod(label(bool.vddr))
sentencia → if(bool) sentencia1 else sentencia2	bool.vddr = nuevaEtq() bool.fls = nuevaEtq() sentencia1.sig = sentencia.sig sentencia2.sig = sentencia.sig genCod(label(bool.vddr)) genCod('goto' sentencia.sig) genCod(label(bool.fls))
sentencia → while(bool) sentencia1	sentencia1.sig = nuevaEtq() bool.vddr = nuevaEtq() bool.fls = sentencia.sig genCod(label(sentencia1.sig)) genCod(label(bool.vddr)) genCod('goto' sentencia1.sig)
sentencia → do sentencia1 while(bool)	bool.vddr = nuevaEtq() bool.fls = sentencia.sig sentencia1.sig = nuevaEtq() genCod(label(bool.vrdd)) genCod(label(sentencia1.sig))
sentencia → break ;	genCod(goto sentencia.sig)
sentencia → bloque	bloque.sig = sentencia.sig
sentencia → return exp ;	ListaRetorno.agregar(exp.tipo) genCod('return' exp.dir)
sentencia → return;	ListaRetorno.agregar(void) genCod('return')
sentencia → switch(bool) { casos }	casos.etqprueba = nuevaEtq() genCode('goto' casos.etqprueba) casos.sig = sentencia.sig casos.id = bool.dir genCode(label(casos.etqprueba)) genCode(casos.prueba)
sentencia → scan parte izquierda	genCod("print" exp.dir);
sentencia →print exp ;	genCod("scan" parte_izquierda.dir)
casos → caso caso 1	casos1.sig = casos.sig caso.sig = casos.sig casos.prueba = caso.prueba k casos1.prueba
casos → ε	casos.prueba =VACIA
casos → predeterminado	casos.prueba = predeterminado.prueba predeterminado.sig = casos.sig
caso → case numero: instrucciones	caso.inicio = nuevaEtq() instrucciones.sig = caso.sig caso.prueba = genCod(if caso.id '=='

	numero.lexval 'goto' caso.inicio) genCode(label(caso.inicio))
predeterminado→ default: instrucciones	predeterminado.inicio = nuevaEtq() instrucciones.sig = predeterminado.sig predeterminado.prueba = genCod('goto' predeterminado.inicio) genCode(label(predeterminado.inicio))
bool →comb bool'	comb.vddr = bool.vddr comb.fls =nuevoIndice() bool'.tipoH = comb.tipo bool'.lista_indices = nuevaListaIndices() bool'.lista_indices.agregar(comb.fls) bool.tipo = bool'.tipoS genCod(label(comb.fls))
bool'→|| comb bool' 1	Si equivalentes(bool'.tipoH, comb.tipo) Entonces comb.vddr = bool.vddr comb.fls =nuevoIndice() bool'$_1$ .tipoH = comb.tipo bool'$_1$ .vddr = bool.vddr bool'$_1$ .fls = bool.fls bool'$_1$.lista_indices = bool'$_1$.lista_indices bool'$_1$.lista_indices.agregar(comb.fls) bool'.tipoS = bool'$_1$.tipoS genCod(label(bool1 .fls)) Sino error("Tipos incompatibles") FinSi
bool'→ε	reemplazarIndices(bool'.lista_indices, bool'.fls, cuadruplas) bool'.tipoS = int
comb →igualdad comb'	igualdad.vddr = nuevoIndice() igualdad.fls =comb.fls comb'.tipoH = igualdad.tipo

	comb'.lista_indices = nuevaListaIndices() comb'.lista_indices.agregar(igualdad .vddr) comb.tipo = comb'.tipoS genCod(label(igualdad.vddr))
comb' →&& igualdad comb'|ε	Si equivalentes(comb'.tipoH, igualdad.tipo) Entonces igualdad.vddr = nuevoIndice() igualdad.fls = comb.fls comb'$_1$ .tipoH = comb.tipo comb'$_1$ .vddr = bool.vddr comb'$_1$ .fls = bool.fls comb'$_1$.lista_indices = bool'$_1$.lista_indices comb'$_1$.lista_indices.agregar(igualdad.vddr) comb'.tipoS = comb'$_1$.tipoS genCod(label(igualdad.vddr)) Sino error("Tipos incompatibles") FinSi
comb' →ε	reemplazarIndices(comb'.lista_indices, comb'.vddr, cuadruplas) comb'.tipoS = int
Igualdad →rel igualdad'	igualdad'.vddr = igualdad.vddr igualdad'.fls = igualdad.fls igualdad'.dirH = rel.dir igualdad'.tipoH = rel.tipo igualdad.dir = igualdad'.dirS igualdad.tipo = igualdad'.tipoS
igualdad'→==rel igualdad'1	Si equivalentes(igualdad'.tipoH, rel.tipo) Entonces igualdad'$_1$.tipoH = rel.tipo igualdad'$_1$.dirH = rel.dir igualdad'.dirS=nuevaTemporal(); tipoTemp = maximo(igualdad'.tipoH, rel.tipo) d1 = ampliar(igualdad'.dirH, igualdad'.tipoH, tipoTemp) d2 = ampliar(rel.dir, rel.tipo, tipoTemp) genCod(igualdad'.dirS '=' d1 '==' d2) genCod('if' igualdad'.dirS 'goto' igualdad'.vddr) genCod('goto' igualdad'.fls) Sino

	error(”Tipos incompatilbes”) FinSi igualdad’.tipoS = igualdad’$_1$tipoS --igualdad’.tipoS = int(probable)
igualdad’⟶ != rel igualdad’1	Si equivalentes(igualdad’.tipoH, rel.tipo) Entonces igualdad’$_1$.tipoH = rel.tipo igualdad’$_1$.dirH = rel.dir igualdad’.dirS=nuevaTemporal(); tipoTemp = maximo(igualdad’.tipoH, rel.tipo) d1 = ampliar(igualdad’.dirH, igualdad’.tipoH, tipoTemp) d2 = ampliar(rel.dir, rel.tipo, tipoTemp) genCod(igualdad’.dirS ’=’ d1 ’!=’ d2) genCod(’if’ igualdad’.dirS ’goto’ igualdad’.vddr) genCod(’goto’ igualdad’.fls) Sino error(”Tipos incompatilbes”) FinSi igualdad’.tipoS = igualdad’$_1$tipoS --igualdad’.tipoS = int(probable)
igualdad’⟶ε	igualdad’.tipoS = igualdad’.tipoH igualdad’.dirS = igualdad’.dirH
rel⟶exp rel’	 rel’.vddr = rel.vdrr rel’.fls =rel.fls rel’.tipoH = exp.tipoS rel’.dirH = exp.dir rel.tipo = rel’.tipoS rel.dir = rel’.dirS
rel’⟶<exp	Si equivalentes(exp.tipoS, rel’.tipoH) Entonces rel’.tipoS = int rel’.dirS = nuevaTemporal() genCod(rel’.dirS ’=’ ’<’ exp.dir) genCod(’if’ rel’.dirS ’goto’ rel’.vddr) genCod(’goto’ rel’.fls) Sino error(”Tipos incompatibles”) FinSi
rel’⟶<=exp	Si equivalentes(exp.tipo, rel’.tipoS) Entonces rel’.tipoS = int rel’.dirS = nuevaTemporal() genCod(rel’.dirS ’=’ ’<=’ exp.dir) genCod(’if’ rel’.dirS ’goto’ rel’.vddr) genCod(’goto’ rel’.fls) Sino error(”Tipos incompatibles”) FinSi
rel’⟶>=exp	Si equivalentes(exp.tipo, rel’.tipo) Entonces rel’.tipoS = exp.tipoS rel’.dirS = nuevaTemporal() genCod(rel’.dirS ’=’ ’>=’ exp.dir) genCod(’if’ rel’.dirS ’goto’ rel’.vddr) genCod(’goto’ rel’.fls)

	Sino error(”Tipos incompatilbes”) FinSi
rel’→>exp	Si equivalentes(exp.tipo, rel’.tipo) Entonces rel’.tipo = int rel’.dirS = nuevaTemporal() genCod(rel’.dirS ’=’ ’>’ exp.dir) genCod(’if’ rel’.dirS ’goto’ rel’.vddr) genCod(’goto’ rel’.fls) Sino error(”Tipos incompatibles”) FinSi
rel’→ε	rel’.tipoS = rel’.tipoH rel’.dirS = rel’.dirH
exp → term exp’	exp’.tipoH = term.tipoS exp’.dirH= term.dirS exp.dir = exp’.dirS exp.tipo = exp’.tipoS
exp’ → +term exp’ 1	si equivalentes(exp’.tipoH,term.tipo) Entonces exp’1.tipoH = term.tipo exp’1.dirH = term.dir exp’ .tipoS= maximo(exp’$_1$.tipoS,term.tipoS) exp’ .dirS = nuevaTemporal() d1 = ampliar(exp’.dirH, exp’.tipoH,exp’ .tipoS) d2 = ampliar(term.dir, term.tipo, exp’ .tipoS) genCod(exp’ .dirS || ‘=’ || d1 ||’+’|| d2l) exp’.tipoS = exp’$_1$.tipoS exp’.dirS = exp’$_1$.dirS Sino error(“No son compatibles”) FinSi
exp’ →-term exp’ 1	si equivalentes(exp’.tipoH,term.tipo) Entonces exp’1.tipoH = term.tipo exp’1.dirH = term.dir exp’ .tipoS= maximo(exp’$_1$.tipoS,term.tipoS) exp’ .dirS = nuevaTemporal() d1 = ampliar(exp’.dirH, exp’.tipoH,exp’ .tipoS) d2 = ampliar(term.dir, term.tipo, exp’ .tipoS) genCod(exp’ .dirS || ‘=’ || d1 ||’-’|| d2l) exp’.tipoS = exp’$_1$.tipoS exp’.dirS = exp’$_1$.dirS Sino error(“No son compatibles”) FinSi
exp’ → ε	exp’.tipoS= exp’.tipoH exp’.dirS = exp’.dirH
term→unario term’	term’.tipoH = unario.tipo term’.dirH = unario.dir

	term.dir=term'.dirS term.tipo=term'.tipoS
term' →*unario term'1	Si equivalentes(term'.tipoH, unario.tipo) entonces term'1.tipoH = unario.tipo term'1.dirH = unario.dir term'.tipoS = maximo(term'.tipoH, unario.tipo) term'.dirS = nueva Temporal() d1 = ampliar(term'.dirH, term'.tipoH,term' .tipoS) d2 = ampliar(unario.dir, unario.tipo, term' .tipoS) genCodigo(term'.dirS || '=' || d1 || '*' || d2) term'.tipoS = $term'_1$.tipoS Sino error("los tipos no son compatibles") FinSi
term' → /unario term'1	Si equivalentes(term'.tipoH, unario.tipo) entonces term'1.tipoH = unario.tipo term'1.dirH = unario.dir term'.tipoS = maximo(term'.tipoH, unario.tipo) term'.dirS = nueva Temporal() d1 = ampliar(term'.dirH, term'.tipoH,term' .tipoS) d2 = ampliar(unario.dir, unario.tipo, term' .tipoS) genCodigo(term'.dirS || '=' || d1 || '/' || d2) term'.tipoS = $term'_1$.tipoS Sino error("los tipos no son compatibles") FinSi
term' →%unario term' 1	Si equivalentes(term'.tipoH, unario.tipo) entonces term'1.tipoH = unario.tipo term'1.dirH = unario.dir term'.tipoS = int term'.dirS = nueva Temporal() genCodigo(term'.dirS || '=' || term'.dirH || '%' || unario.dir) term'.tipoS = $term'_1$.tipoS Sino error("los tipos no son compatibles") FinSi
term' →ε	term'.tipoS = term.tipoH term'.dirS = term.dirH
unario → !unario1	unario.dir = nuevaTemporal() unario.tipo = unario1.tipo genCod(unario.dir '=' '!' unario1.dir)
unario →- unario1	unario.dir = nuevaTemporal() unario.tipo = unario1.tipo

	genCod(unario.dir '=' '-' unario1.dir)
unario →factor	unario.dir = factor.dir unario.tipo = factor.tipo
factor → (bool)	factor.tipo = bool.tipo factor.dir = bool.dir
factor → numero	factor.dir = numero.lexval factor.tipo = numero.lextipo
factor →cadena	TablaCadenas.agregar(cadena.lexval) factor.dir = TablaCadenas.getUltimaPos() factor.tipo = cadena
factor → true	factor.dir = 'true' factor.tipo = int
factor → false	factor.dir = 'false' factor.tipo = int
factor → Id factor'	factor'.base=id.lexval factor.dir = factor'.dir factor.tipo = factor'.tipo
factor'→ localizacion	Localizacion.base = factor'.base factor'.dir = nuevaTemporal() factor'.tipo = localización.tipo genCod(factor'.dir '=' factor'.base '[localizacion.dir']')

factor'→ (parametros)	Si PilaTS.fondo().buscar(factor'.base) Entonces Si PilaTS.fondo().getVar(factor'.base) = 'func' Entonces Si equivalenteListas(PilaTS.fond().getArgs(factor'.base), parametores.lista) Entonces factor.tipo = PilaTS.top().getTipo(id) factor.dir = nuevaTemporal() genCod(factor.dir '=' 'call' id ',' parametros.lista.tam) Sino error("El número o tipo de parámetros no coincide") FinSi Sino error("El id no es una función") FinSi Sino error("El id no está declarado") FinSi
factor'→ ε	factor'.dir =factor'.base factor'.tipo=PilaTS.top().getTipo(factor'.dir)
parametros → lista param	parametros.lista = lista parametros.lista
parametros → ε	parametros.lista = NULO
Lista_param→ bool	bool.tipoH=Lista_param.tipo
lista_param'→,bool lista_param'1	lista_param'1.lista=lista_param'.lista
lista_param'→ ε	
localizacion→[bool] localizacion'	Si PilaTS.top().buscar(localizacion.base) Entonces Si bool.tipo = int Entonces tipoTmp = PilaTS.top().getTipo(localizacion.base) Si PilaTT.top().getNombre(tipoTmp)='array' Entonces localizacion'.tipo = PilaTT.top().getTipoBase(tipoTmp) localizacion'.dir = nuevaTemporal() localizacion'.tam = PilaTT.top().getTam(localizacion'.tipo) genCod(localizacion.dir '=' bool.dir '*' localizacion.tam) localizacion.dir = localizacion'.dirS localizacion.tipo = localizacion'.tipoS Sino error("El id no es un arreglo") FinSi Sino error("El indice del arreglo debe ser entero")

	FinSi Sino error(”El id no está declarado”) FinSi
localizacion’→ [bool] localizacion’	Si bool.tipo = int Entonces Si PilaTT.top.getNombre(localizacion’.tipo)=’array’ Entonces localizacion’$_1$.tipo = PilaTT.top().getTipoBase(localizacion’.tipo) dirTmp = nuevaTemporal() localizacion’$_1$.dir = nuevaTemporal() localizacion’$_1$.tam = PilaTT.top().getTam(localizacion’.tipo) genCod(dirTmp’=’bool.dir’*’ localizacion’$_1$.tam genCod(localizacion’$_1$.dir ’=’localizacion’.dir’+’ dirTmp) localizacion’.dir = localizacion’$_1$.dirS localizacion’.tipo = localizacion’$_1$.tipoS Sino error(”El id no es un arreglo”) FinSi Sino error(”El indice del arreglo debe ser entero”) FinSi
localizacion’→ ε	localizacion’.dirS=localizacion’.dir localizacion’.tipoS=localizacion.tipo

Esquema de traducción (solo es con una columna)

programa→
{PilaTS.push(nuevaTablaTS())
PilaTT.push(nuevaTablaTT())
dir = 0}
declaraciones funciones

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

declaraciones → ε

tipo → basico {compuesto.base = basico.tipo } compuesto {tipo.tipo = compuesto.tipo}

basico → int {basico.tipo = int }

basico →float {basico.tipo = float }

basico → char{basico →char}

basico → double {basico.tipo = double }
basico → void{basico.tipo = void}

compuesto → [numero] {compuesto1.base = compuesto.base } compuesto1 {compuesto.tipo = PilaTT.top().insertar(”array”, num.val, compuesto1.tipo) }

compuesto → ε {compuesto.tipo = compuesto.base }

lista_var→ id{lista var’.tipo = lista var.tipo
Si ! PilaTS.top().buscar(id) Entonces
  PilaTS.top().insetar(id, lista var.tipo, dir, ”var”, NULO)

dir = dir + PilaTT.top().getTam(tipo.tipo)
Sino
error(”El id no esta declarado”)
FinSi } lista_var’

lista_var’→, id {
Si ! PilaTS.top().buscar(id) Entonces
PilaTS.top().insetar(id, lista var.tipo, dir, ”var”, NULO)
dir = dir + PilaTT.top().getTam(tipo.tipo)
Sino
error(”El id no esta declarado”) ´
FinSi } lista_var’

lista_var’→ ε{lista_var’.tipo = lista_var’.base }

funciones →{ListaRetorno = NULO --nueva lista
PilaTS.push(nuevaTablaSimbolos) PilaTT.push(nuevaTablaTipos)
PilaDir.push(dir)
dir = 0 } func tipo idSi ! PilaTS.fondo().buscar(id) Entonces {
(argumentos)
genCod(label(id))
bloque.sig = nuevaEtq()
bloque
Si equivalentesLista(ListaRetorno, tipo.tipo) Entonces --

PilaTS.fondo().insetar(id, tipo.tipo, –, ’func’, argumentos.lista)

genCod(label(bloque.sig))
Sino
error(”Los tipos de retorno no coinciden con el tipo de la funcion”)
FinSi
Sino
error(”El id no esta declarado”)
FinSi

PilaTS.pop()
PilaTT.pop()
dir = PilaDir.pop() } funciones

argumentos→ lista args {argumentos.lista = lista args.lista}

argumentos→ ε{**ar**gumentos.lista = NULO }

lista_args→tipo id{lista_args’.listaH = nuevaLista()
lista_args’.listaH.agregar(tipo.tipo)} lista_args’ {Lista_args.lista = lista_args’.listaS} (camba semantico)

lista_args’→, tipo id {lista_args´1.listaH = lista_args’.listaH
lista_args’1.listaH.agregar(tipo.tipo)}lista_args’1 {lista_args´.listaS = lista_args’1.listaS}

lista_args’→ε{lista_args’.listaS = Lista_args’.listaH}

bloque → { declaraciones instrucciones }

instrucciones →{sentencia.sig = nuevaEtq()}sentencia{genCod(label(sentencia.sig))} instrucciones‘

instrucciones’ →sentencia {instrucciones’1.sig = nuevaEtq() } instrucciones‘1 {genCod(label(instrucciones’1.sig))}

instrucciones’ →ε

parte_izquierda→**id{**parte_izquierda’.base = id.lexval **}** parte_izquierda’ {parte_izquierda.dir = parte_izquierda’.dir’.tipo

parte_izqiuerda.tipo = parte_izquierda}

parte_izquierda'→{localizacion.base =parte_izquierda'.base}localizacion {parte_izquierda'.dir = localizacion.dir

parte_izqiuerda'.tipo = localizacion.tipo}

parte_izquierda'→ε{Si PilaTS.top().buscar(parte_izquierda'.base) Entonces

parte_izquierda'.dir = parte_izquierda'.base

parte_izuierda'.tipo = PilaTS.top().getTipo(parte_izq'.dir)

Sino

error("El id no está declarado")

FinSi}

sentencia → parte_izquierda= bool {Si equivalentes(parte_izquierda.tipo, bool.tipo) Entonces
d1 = reducir(bool.dir, bool.tipo,parte_izquierda.tipo)
genCod(parte_izquierda.dir '=' d1)
Sino
error("Tipos incompatibles")
FinSi };

sentencia → if({bool.vddr = nuevaEtq()
bool.fls = sentencia.sig }bool) {sentencia1.sig = sentencia.sig
genCod(label(bool.vddr)) }sentencia1

sentencia → if({bool.vddr = nuevaEtq()
bool.fls = nuevaEtq()
genCod(label(bool.vddr))
genCod(label(bool.fls))} bool) {sentencia1.sig = sentencia.sig }sentencia1 else {sentencia2.sig =
sentencia.sig}sentencia2 {genCod('goto' sentencia.sig) }

sentencia → while({bool.vddr = nuevaEtq()
bool.fls = sentencia.sig
genCod(label(bool.vddr)) } bool){sentencia1.sig = nuevaEtq()
genCod(label(sentencia1.sig))
genCod('goto' sentencia1.sig) } sentencia1

sentencia → do {sentencia1.sig = nuevaEtq()
genCod(label(sentencia1.sig))}sentencia1 while({bool.vddr = nuevaEtq()
bool.fls = sentencia.sig
genCod(label(bool.vrdd)) }bool)
sentencia → break ;{genCod(goto sentencia.sig) }

sentencia →{bloque.sig = sentencia.sig} bloque

```
sentencia → return exp ; {ListaRetorno.agregar(exp.tipo)
                          genCod('return' exp.dir)}
sentencia → return;{ListaRetorno.agregar(void)
                    genCod('return') }
sentencia → switch( bool ) { {casos.etqprueba = nuevaEtq()
                          genCode('goto' casos.etqprueba)
                          casos.sig = sentencia.sig
                          casos.id = bool.dir
                          genCode(label(casos.etqprueba))
                          genCode(casos.prueba) }casos }
sentencia → scan parte izquierda{genCod("scan" parte_izquierda.dir)}

sentencia →print exp ; {genCod("print" exp.dir);}

casos → caso{casos1.sig = casos.sig } caso 1 {caso.sig = casos.sig
casos.prueba = caso.prueba || casos1.prueba }

casos → ε {casos.prueba =VACIA}

casos →{predeterminado.sig = casos.sig } predeterminado {casos.prueba = predeterminado.prueba }

caso → case numero:{instrucciones.sig = caso.sig } instrucciones {caso.inicio = nuevaEtq()
                          caso.prueba = genCod(if caso.id '==' numero.lexval 'goto' caso.inicio)
                          genCode(label(caso.inicio))
                          }
predeterminado→ default:{instrucciones.sig = predeterminado.sig } instrucciones {predeterminado.inicio = nuevaEtq()
                          predeterminado.prueba = genCod('goto' predeterminado.inicio)
                          genCode(label(predeterminado.inicio))}

bool →{comb.vddr = bool.vddr
comb.fls =nuevoIndice()}comb{bool'.tipoH = comb.tipo

bool'.lista_indices = nuevaListaIndices()

bool'.lista_indices.agregar(comb.fls)

} bool'{bool.tipo = bool'.tipoS}(cambio semantico)

bool'→|| {comb.vddr = bool.vddr
        comb.fls =nuevoIndice()}comb {Si equivalentes(bool'.tipoH, comb.tipo) Entonces

                          bool'1 .tipoH = comb.tipo

                          bool'1 .vddr = bool.vddr

                          bool'1 .fls = bool.fls

                          bool'1.lista_indices =  bool'1.lista_indices

                          bool'1.lista_indices.agregar(comb.fls)

                          genCod(label(bool'1 .fls))

                          } bool' 1{bool'.tipoS = bool'1.tipoS

                          Sino

                                  error("Tipos incompatibles")

                          FinSi}
bool'→ε{reemplazarIndices(bool'.lista_indices, bool'.fls, cuadruplas)
```

```
bool'.tipoS = int}

comb →{igualdad.vddr = nuevoIndice()
igualdad.fls =comb.fls
 genCod(label(igualdad.vddr))}igualdad {

comb'.tipoH = igualdad.tipo

comb'.lista_indices = nuevaListaIndices()

comb'.lista_indices.agregar(igualdad .vddr)}comb'{comb.tipo = comb'.tipoS
                                                  }
comb' →&&{igualdad.vddr = nuevoIndice()
                    igualdad.fls = comb.fls
                    igualdad.vddr = comb.vddr
} igualdad {{Si equivalentes(comb'.tipoH, igualdad.tipo) Entonces
                    comb'1 .tipoH = comb.tipo
                              comb'1 .vddr = bool.vddr
                              comb'1 .fls = bool.fls
                              comb'1.lista_indices =  bool'1.lista_indices
                              comb'1.lista_indices.agregar(igualdad.vddr)

}comb'1{                      comb'.tipoS = comb'1.tipoS
                              genCod(label(igualdad.vddr))
                    Sino
                              error("Tipos incompatibles")
                    FinSi

                                        }

comb' →ε{reemplazarIndices(comb'.lista_indices, comb'.vddr, cuadruplas)
comb'.tipoS = int}

Igualdad →rel {igualdad'.vddr = igualdad.vddr
                    igualdad'.fls = igualdad.fls
                    igualdad'.dirH = rel.dir
                    igualdad'.tipoH = rel.tipo}igualdad'{igualdad.dir = igualdad'.dirS
                                                        igualdad.tipo = igualdad'.tipoS}
igualdad'→==rel{Si equivalentes(igualdad'.tipoH, rel.tipo) Entonces
                     igualdad'1.tipoH = rel.tipo
                     igualdad'1.dirH = rel.dir
                    }igualdad'1{
                     tipoTemp = maximo(igualdad'.tipoH, rel.tipo)
                          d1 = ampliar(igualdad'.dirH, igualdad'.tipoH, tipoTemp)
                         d2 = ampliar(rel.dir, rel.tipo, tipoTemp)
                        igualdad'.dirS=nuevaTemporal();
                                    genCod(igualdad'.dirS '=' d1 '==' d2)
                                         genCod('if' igualdad'.dirS 'goto'
                                                        igualdad'.vddr)
                                         genCod('goto' igualdad'.fls)
                                    Sino
                                         error("Tipos incompatilbes")
                                     FinSi
                                  igualdad'.tipoS = igualdad'1tipoS}
igualdad'→ !=  rel {Si equivalentes(igualdad'.tipoH, rel.tipo) Entonces
        igualdad'1.tipoH = rel.tipo
        igualdad'1.dirH = rel.dir
            }igualdad'1{
```

```
        igualdad'.dirS=nuevaTemporal();
        tipoTemp = maximo(igualdad'.tipoH, rel.tipo)
        d1 = ampliar(igualdad'.dirH, igualdad'.tipoH, tipoTemp)
       d2 = ampliar(rel.dir, rel.tipo, tipoTemp)

        genCod(igualdad'.dirS '=' d1 '!=' d2)
        genCod('if' igualdad'.dirS 'goto'
                          igualdad'.vddr)
        genCod('goto' igualdad'.fls)
Sino
        error("Tipos incompatilbes")
 FinSi
igualdad'.tipoS = igualdad'1.tipoS}
 igualdad'⟶ε{igualdad'.tipoS = igualdad'.tipoH
                    igualdad'.dirS = igualdad'.dirH}

rel→exp
{rel'.tipoH=exp.tipo
rel'.dirH= exp.dir
rel'.vddr = rel.vddr
rel'.fls = rel.fls}
rel'
{rel.tipo=rel',tipoS
rel.dir=rel'.dirS}

rel'→<=exp{Si equivalentes(rel'.tipoH, exp.tipo) Entonces
   rel'.tipo = int
   rel'.dir= nuevaTemporal()
   tipoTemp = maximo(rel'.tipoH,exp.tipo)

   d1 = ampliar(rel'.dirH,exp.tipoH, tipoTemp)
   d2 = ampliar(rel'.dirH, exp.tipo, tipoTemp)
   genCod(rel'.dirH '=' d1.dir '<=' d2.dir)
   genCod('if' rel'.dirH 'goto' rel'.vddr)
   genCod('goto' rel'.fls)
Sino
     error("Tipos incompatilbes")
FinSi}

rel'→>=exp {Si equivalentes(rel'.tipoH, exp.tipo) Entonces
   rel'.tipo = int
   rel'.dir= nuevaTemporal()
   tipoTemp = maximo(rel'.tipoH,exp.tipo)

   d1 = ampliar(rel'.dirH,exp.tipoH, tipoTemp)
   d2 = ampliar(rel'.dirH, exp.tipo, tipoTemp)
   genCod(rel'.dirH '=' d1.dir '>=' d2.dir)
   genCod('if' rel'.dirH 'goto' rel'.vddr)
   genCod('goto' rel'.fls)
Sino
     error("Tipos incompatilbes")
FinSi}

rel'→>exp {Si equivalentes(rel'.tipoH, exp.tipo) Entonces
   rel'.tipo = int
   rel'.dir= nuevaTemporal()
   tipoTemp = maximo(rel'.tipoH,exp.tipo)

   d1 = ampliar(rel'.dirH,exp.tipoH, tipoTemp)
   d2 = ampliar(rel'.dirH, exp.tipo, tipoTemp)
   genCod(rel'.dirH '=' d1.dir '>' d2.dir)
   genCod('if' rel'.dirH 'goto' rel'.vddr)
   genCod('goto' rel'.fls)
Sino
     error("Tipos incompatilbes")
```

FinSi}

rel'→rel'.tipoS = rel'.tipoH
rel'.dirS = rel'.dirH

rel'→ε{rel'.tipoS = rel'.tipoH
rel'.dirS = rel'.dirH}

exp → term{exp'.tipoH = term.tipo
exp'.dirH= term.dir} exp'{exp.dir = exp'.dirS
exp.tipo = exp'.tipoS}

exp' → +term{si equivalentes(exp'.tipoH,term.tipo) Entonces
exp'1.tipoH = term.tipo
exp'1.dirH = term.dir} exp' 1{ exp' .tipoS= maximo(exp'$_1$.tipoS,term.tipoS)
exp' .dirS = nuevaTemporal()

d1 = ampliar(exp'.dirH, exp'.tipoH,exp' .tipoS)
d2 = ampliar(term.dir, term.tipo, exp' .tipoS)
genCod(exp' .dirS || '=' || d1 ||'+'|| d2l)
exp'.tipoS = exp'$_1$.tipoS
exp'.dirS = exp'$_1$.dirS
Sino
error("No son compatibles")
FinSi
}
exp' →-term{si equivalentes(exp'.tipoH,term.tipo) Entonces
exp'1.tipoH = term.tipo
exp'1.dirH = term.dir} exp' 1{exp' .tipoS= maximo(exp'$_1$.tipoS,term.tipoS)
exp' .dirS = nuevaTemporal()

d1 = ampliar(exp'.dirH, exp'.tipoH,exp' .tipoS)
d2 = ampliar(term.dir, term.tipo, exp' .tipoS)
genCod(exp' .dirS || '=' || d1 ||'-'|| d2l)
exp'.tipoS = exp'$_1$.tipoS
exp'.dirS = exp'$_1$.dirS
Sino
error("No son compatibles")
FinSi
}
exp' → ε{exp'.tipoS= exp'.tipoH
exp'.dirS = exp'.dirH}

term→unario{term'.tipoH = unario.tipo
term'.dirH = unario.dir} term'{term.dir=term'.dirS
term.tipo=term'.tipoS}

term' →*unario{Si equivalentes(term'.tipoH, unario.tipo) entonces
term'1.tipoH = unario.tipo
term'1.dirH = unario.dir
} term'1{
term'.tipoS = maximo(term'.tipoH, unario.tipo)
term'.dirS = nueva Temporal()
d1 = ampliar(term'.dirH, term'.tipoH,term' .tipoS)
d2 = ampliar(unario.dir, unario.tipo, term' .tipoS)
genCodigo(term'.dirS || '=' || d1 || '*' || d2)

```
                    term'.tipoS = term'1.tipoS
            Sino
                     error("los tipos no son compatibles")
            FinSi
}
term' → /unario{Si equivalentes(term'.tipoH, unario.tipo) entonces
                        term'1.tipoH = unario.tipo
                        term'1.dirH = unario.dir} term'1{
                        term'.tipoS = maximo(term'.tipoH, unario.tipo)
                        term'.dirS = nueva Temporal()
                        d1 = ampliar(term'.dirH, term'.tipoH,term' .tipoS)
                        d2 = ampliar(unario.dir, unario.tipo, term' .tipoS)
                        genCodigo( term'.dirS || '=' || d1 || '/' || d2)
                         term'.tipoS = term'1.tipoS
                    Sino
                        error("los tipos no son compatibles")
                    FinSi}
term' →%unario{Si equivalentes(term'.tipoH, unario.tipo) entonces
                        term'1.tipoH = unario.tipo
                        term'1.dirH = unario.dir} term' 1{
                        term'.tipoS = int
                        term'.dirS = nueva Temporal()
                        genCodigo( term'.dirS || '=' || term'.dirH || '%' || unario.dir)
                        term'.tipoS = term'1.tipoS
                    Sino
                        error("los tipos no son compatibles")
                    FinSi}

term' →ε{term'.tipoS = term.tipoH
term'.dirS = term.dirH}
unario → !unario1 {unario.dir = nuevaTemporal()
                   unario.tipo = unario1.tipo
                   genCod(unario.dir '=' '!' unario1.dir)  }
unario →− unario1{unario.dir = nuevaTemporal()
                  unario.tipo = unario1.tipo
                  genCod(unario.dir '=' '-' unario1.dir) }
unario →factor{unario.dir = factor.dir
               unario.tipo = factor.tipo }

factor → (bool)  {factor.tipo = bool.tipo

factor.dir = bool.dir }



factor → numero {factor.dir = numero.lexval

factor.tipo = numero.lextipo}



factor →cadena {TablaCadenas.agregar(cadena.lexval)

factor.dir = TablaCadenas.getUltimaPos()

factor.tipo = cadena}
```

factor → true {factor.dir = 'true'

factor.tipo = int }

factor → false {factor.dir = 'false'

factor.tipo = int}

factor →Id {factor'.base=id.lexval}factor'{factor.dir = factor'.dir
factor.tipo = factor'.tipo}

factor'→{Localizacion.base = factor'.base} localizacion {factor'.dir = nuevaTemporal()

factor'.tipo = localización.tipo

genCod(factor'.dir '=' factor'.base '[localizacion.dir']')}

factor'→ Si PilaTS.fondo().buscar(factor'.base) Entonces
Si PilaTS.fondo().getVar(factor'.base) = 'func' Entonces
Si equivalenteListas(PilaTS.fond().getArgs(factor'.base) ,parametores.lista) Entonces
(parametros) {
factor'.dir = nuevaTemporal()
factor'.tipo = PilaTS.top().getTipo(id)
genCod(factor.dir '=' 'call' id ',' parametros.lista.tam)
Sino
error("El número o tipo de parámetros no coincide")
FinSi
Sino
error("El id no es una función")
FinSi
Sino
error("El id no está declarado")
FinSi

}

factor'→ ε{factor'.dir =factor'.base

factor'.tipo=PilaTS.top().getTipo(factor'.dir)}

parametros → lista_param {parametros.lista = lista_parametros.lista }

parametros → ε {parametros.lista = NULO }

Lista_param→ {bool.tipoH=Lista_param.tipo}bool lista_param'

lista_param'→,bool lista_param'1{lista_param'1.lista=lista_param'.lista}

lista_param'→ ε

localizacion→[bool] {Si PilaTS.top().buscar(localizacion.base) Entonces

Si bool.tipo = int Entonces

tipoTmp = PilaTS.top().getTipo(localizacion.base)

Si PilaTT.top().getNombre(tipoTmp)='array' Entonces

localizacion'.tipo = PilaTT.top().getTipoBase(tipoTmp)

localizacion'.dir = nuevaTemporal()

localizacion'.tam =

PilaTT.top().getTam(localizacion'.tipo)

genCod(localizacion.dir '=' bool.dir '*' localizacion.tam)}localizacion'{ localizacion.dir = localizacion'.dirS

localizacion.tipo = localizacion'.tipoS

Sino

error("El id no es un arreglo")

FinSi

Sino

error("El indice del arreglo debe ser entero")

FinSi

Sino

error("El id no está declarado")

FinSi

}

localizacion'→ [bool]{Si bool.tipo = int Entonces

Si PilaTT.top.getNombre(localizacion'.tipo)='array' Entonces

localizacion'$_1$.tipo =

PilaTT.top().getTipoBase(localizacion'.tipo)

dirTmp = nuevaTemporal()

localizacion'$_1$.dir = nuevaTemporal()

localizacion'$_1$.tam =

PilaTT.top().getTam(localizacion'.tipo)

genCod(dirTmp'='bool.dir'*' localizacion'$_1$.tam

genCod(localizacion'$_1$.dir '='localizacion'.dir'+'

dirTmp)

} localizacion'1{ localizacion'.dir = localizacion'$_1$.dirS

localizacion'.tipo = localizacion'$_1$.tipoS

Sino

error("El id no es un arreglo")

FinSi

Sino

error("El indice del arreglo debe ser entero")

FinSi

}

localizacion'→ ε{localizacion'.dirS=localizacion'.dir

localizacion'.tipoS=localizacion.tipo}