UNIVERSIDAD NACIONAL

AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Compiladores

Espinosa González Isaac
Montoya Pérez Héctor
Soto Vázquez Patricia
Pérez Dublán Juan Pablo

Esquema de traducción

PRODUCCIÓN	ESQUEMA
programa → declaraciones funciones	programa → {STS.push(newTS()); STT.push(newTT()); dir = 0} declaraciones funciones {programa.codigo = funciones.codigo}
declaraciones → tipo lista var; declaraciones	declaraciones→{typeGBL = tipo.type} tipo lista var; declaraciones
declaraciones → tipo registro lista var; declaraciones declaraciones →ε	declaraciones → {typeGBL= tipo registro.type}tipo registro lista var; declaraciones declaraciones →ε
tipo_registro → estructura inicio declaraciones fin	tipo registro→{STS.push(newTS()); STT.push(newTT()); SDir.push(dir) ;dir = 0; } estructura inicio declaraciones fin { 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	tipo → base {baseGBL = base.base} tipo_arreglo {tipo.type = tipo_arreglo.type}
base → ent	base → ent {base.base = STT.getTop().getType(’ent’)}
base → real	base → real {base.base = STT.getTop().getType(’real’)}
base → dreal	base → dreal {base.base = STT.getTop().getType(’dreal’)}
base → car	base → car {base.base = STT.getTop().getType(’car’)}
base → sin	base → sin {base.base = STT.getTop().getType(’sin’)}
tipo_arreglo → [num] $tipo\_arreglo_1$	tipo_arreglo→ [num] $tipo\_arreglo_1${ Si num.type = ent Entonces; Si num.dir > 0 Entonces tipo_arreglo.type = STT.getTop().insert(’array’,num, $tipo\_arreglo_1$.tipo); Sino Error(“El tamaño del arreglo no es válido”)

	Fin Si Sino Error(“El tamaño del arreglo no es un número entero”) Fin Si}
tipo_arreglo→ε	tipo_arreglo→ε{tipo_arreglo.type = baseGBL}
lista_var → lista_var$_1$, id	lista_var → lista_var$_1$, 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(“Ya se declaro la variable”) Fin Si}
lista _var → id	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(“Ya se declaró la variable”) Fin Si}

funciones → def tipo
id(argumentos) inicio
declaraciones
sentencias fin funciones
	funciones → def tipo id {
	Si no STS.getGlobal().existe(id) Entonces
	STS.push(newTS())
	STT.push(newTT()) SDir.push(dir) dir = 0
	listaRET = newListRet()
	Sino
	Error(“Ya se declaró la variable”)
	Fin Si
	} (argumentos) inicio declaraciones sentencias fin {
	Si cmpRet(lista_retorno, tipo.type) Entonces
	L = newLabel()
	backpatch(sentencias.nextlist, L)
	genCode(label L)
	STS.pop()
	STT.pop()
	Sino
	Error(“El tipo de retorno no coincide”)
	Fin Si
	} funciones

funciones →ε	funciones →ε
argumentos → lista_arg	argumentos → lista_arg {argumentos.lista = lista_arg.lista; argumentos.num = lista_arg.num}
argumentos → sin	argumentos→sin{argumentos.lista=NULO; argumentos.num=0}
lista _arg → lista_arg$_1$, arg	lista_arg → lista_arg$_1$, arg {lista_arg.lista = lista_arg$_1$.lista; lista_arg.lista.append(arg.type); lista_arg.num = lista_arg$_1$.num + 1}
lista _arg → arg	lista_arg → arg {lista_arg.lista = newList(); lista_arg.lista.append(arg.type); lista arg.num = 1}
arg → tipo_arg id	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("Ya se declaró la variable") Fin Si}
tipo_arg → base param_arr	tipo arg → base {baseGBL = base.base;} param_arr {tipo.type = param arr.type}
param_arr → [ ] param_arr	param_arr → [ ] param_arr{param_arr.type = STT.getTop().insert('array), 0, param_arr$_1$.tipo)}
param_arr →ε	param_arr→ε{param_arr.type = baseGBL}
sentencias→sentencias $_1$ sentencia	sentencias→ sentencias$_1$ {L = newLabel() backpatch(sentencias$_1$.nextlist, L) genCode(label L)} sentencia

sentencia → si e_bool entonces sentencia$_1$ fin	sentencia→ si e_bool entonces sentencia$_1$ fin {L = newLabel() backpatch(e_bool.truelist, L) sentencia.nextlist = combinar(e_bool.falselist, Sentencia$_1$.nextlist) genCode(label L)}
sentencia →si e_bool entonces sentencia$_1$ sino sentencia$_2$ fin	sentencia→ si e_bool entonces sentencia$_1$ sino sentencia$_2$ fin {L_1 = newLabel() L_2 = newLabel() backpatch(e bool.truelist, L_1) backpatch(e bool.falselist, L_2) sentencia.nextlist = combinar(sentencia$_1$.nextlist, sentencia$_2$.nextlist) genCode(label L_1) genCode('goto' sentencia$_1$.nextlist[0]) genCode(label L_2)}
sentencia →mientras e_bool hacer sentencia$_1$ fin	sentencia→mientras e_bool hacer sentencia$_1$ fin {L_1 = newLabel() L_2 = newLabel() backpatch(sentencia$_1$.nextlist,L_1) backpatch(e_bool.truelist, L_2) sentencia.nextlist = e_bool.falselist genCode(label L_1) b genCode(label L_2) genCode('goto' sentencia$_1$.nextlist[0])}
sentencia → hacer sentencia$_1$ mientras e_bool;	sentencia→ hacer {L = newLabel() genCode("label" L)} sentencia$_1$ mientras e_bool; {backpatch(sentencia$_1$.nextlist, L)}
sentencia → segun (variable) hacer casos predeterminado fin	sentencia → segun (variable) hacer casos predeterminado fin {L_1 = newLabel() prueba = combinar(casos.prueba, predeterminado.prueba) backpatch(casos.nextlist, L_2) sustituir("??", variable.dir, prueba)}
sentencia → variable := expresion ;	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 ;	sentencia → escribir expresion ; {gen(”write” expresion.dir)}
sentencia → leer variable ;	sentencia →leer variable ; {gen(”read” variable.dir)}
sentencia → devolver;	sentencia → devolver; {genCode(”return”)}

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

sentencia → inicio $sentencias_1$ fin	sentencia → inicio $sentencias_1$ fin {sentencia.nextlist = $sentencia_1$.nextlist}
casos → $casos_1$caso num: sentencia	casos → $casos_1$caso num:{L = newLabel() /*Indica el inicio del codigo para la sentencia*/ genCode(”label” L) } sentencia {casos.nextlist = combinar(casos.nextlist, $sentencia_1$.nextlist) casos.prueba = $casos_1$.prueba casos.prueba.append(if ”??” ”==” num.dir ”goto” L)}
casos → caso num: sentencia	casos → caso num: {casos.prueba = newCode() L = newLabel() /*Indica el inicio del codigo para la sentencia*/ genCode(”label” L) } sentencia {casos.prueba.append(if ”??” ”==” num.dir ”goto” L) casos.nextlist = sentencia.nextlist}

predeterminado → pred: sentencia	predeterminado → pred: {predeterminado.prueba = newCode() L = newLabel() /*Indica el inicio del codigo para la sentencia*/ genCode(”label” L) } sentencia {predeterminado.prueba.append(”goto” L)}
predeterminado →*ε*	predeterminado →*ε {*pretederminado.prueba = NULO*}*
e_bool → e_bool$_1$ o e_bool$_2$	e_bool → e_bool$_1$ o e_bool$_2${ L = newLabel() backpatch(e bool$_1$.falselist, L) e bool.truelist = combinar(e bool$_1$.truelist, e bool$_2$.truelist) e bool.falselist = e bool$_2$.falselist genCode(label L)}
e _bool → e_bool y e_bool	e _bool → e_bool y e_bool {L = newLabel() backpatch(e bool$_1$.truelist, L) e_bool.truelist = e_bool$_1$.truelist e_bool.falselist = combinar(e_bool$_1$.falselist, e_bool$_2$.falselist) genCode(label L)}
e_bool → no e_bool$_1$	e_bool → no e_bool$_1$ {e_bool.truelist = e_bool$_1$.falselist e_bool.falselist = e_bool.truelist
e_bool → relacional op	e _bool → relacional op {e_bool.truelist = relacional op.truelist e_bool.falselist = relacional op.falselist}
e_bool → verdadero	e _bool → verdadero {index$_0$ = newIndex() e_bool.truelist = newIndexList(index$_0$) genCode(’goto’ index$_0$)}
e_bool → falso	e _bool → falso {index$_0$ = newIndex() e bool.falselist = newIndexList(index$_0$) genCode(’goto’ index$_0$)}
relacional_ op → relacional$_1$ > relacional$_2$	relacional _op → relacional$_1$ > relacional$_2${index$_0$ = newIndex() index$_1$ = newIndex() relacional.truelist = newIndexList(index$_0$) relacional.falselist = newIndexList(index$_1$) genCode(’if’ relacional$_1$.dir > relacional$_2$ ’goto’ index$_0$) genCode(’goto’ index$_1$)
relacional _op → relacional < relacional	relacional_op → relacional < relacional {index$_0$ = newIndex() index$_1$ = newIndex() relacional.truelist = newIndexList(index$_0$) relacional.falselist = newIndexList(index$_1$)

	genCode(’if’ relacional$_1$.dir < relacional$_2$ ’goto’ index$_0$) genCode(’goto’ index$_1$)}
relacional _op → relacional <= relacional	relacional _op → relacional <= relacional{index$_0$ = newIndex() index$_1$ = newIndex() relacional.truelist = newIndexList(index$_0$) relacional.falselist = newIndexList(index$_1$) genCode(’if’ relacional$_1$.dir <= relacional$_2$ ’goto’ index$_0$) genCode(’goto’ index$_1$)}
relacional _op → relacional >= relacional	relacional _op → relacional >= relacional{index$_0$ = newIndex() index$_1$ = newIndex() relacional.truelist = newIndexList(index$_0$) relacional.falselist = newIndexList(index$_1$) genCode(’if’ relacional$_1$.dir >= relacional$_2$ ’goto’ index$_0$) genCode(’goto’ index$_1$)}

relacional_op → relacional <> relacional	relacional_op → relacional <> relaciona{$index_0$ = newIndex() $index_1$ = newIndex() relacional.truelist = newIndexList($index_0$) relacional.falselist = newIndexList($index_1$) genCode('if' $relacional_1$.dir > $relacional_2$ 'goto' $index_0$) genCode('goto' $index_1$)}
relacional_op → relacional <> relacional	relacional_op → relacional <> relacional{$index_0$ = newIndex() $index_1$ = newIndex() relacional.truelist = newIndexList($index_0$) relacional.falselist = newIndexList($index_1$) genCode('if' $relacional_1$.dir > $relacional_2$ 'goto' $index_0$) genCode('goto' $index_1$)}
relacional → expresion	relacional → expresion {relacional.dir = expresion.dir relacional.tipo = expresion.tipo}
expresion → $expresion_1$ + $expresion_2$	expresion → $expresion_1$ + $expresion_2$ {expresion.type = max($expresion_1$.type, $expresion_2$.type) expresion.dir = newTemp() dir_1 = ampliar($expresion_1$.dir, $epxresion_1$.type, expresion.type) dir_2 = ampliar($expresion_2$.dir, $epxresion_2$.type, expresion.type) getCode(expresion.dir '=' dir_1 '+' dir_2)}
expresion → $expresion_1$ − $expresion_2$	expresion → $expresion_1$ − $expresion_2$ {expresion.type = max($expresion_1$.type, $expresion_2$.type) expresion.dir = newTemp() dir_1 = ampliar($expresion_1$.dir, $epxresion_1$.type, expresion.type) dir_2 = ampliar($expresion_2$.dir, $epxresion_2$.type, expresion.type) getCode(expresion.dir '=' dir_1 '-' dir_2)}
expresion → $expresion_1$ * $expresion_2$	expresion → $expresion_1$ * $expresion_2$ {expresion.type = max($expresion_1$.type, $expresion_2$.type) expresion.dir = newTemp() dir_1 = ampliar($expresion_1$.dir, $epxresion_1$.type, expresion.type) dir_2 = ampliar($expresion_2$.dir, $epxresion_2$.type, expresion.type) getCode(expresion.dir '=' dir_1 '*' dir_2)}
expresion → $expresion_1$ / $expresion_2$	expresion → $expresion_1$ / $expresion_2$ {expresion.type = max($expresion_1$.type, $expresion_2$.type) expresion.dir = newTemp() dir_1 = ampliar($expresion_1$.dir, $epxresion_1$.type, expresion.type) dir_2 = ampliar($expresion_2$.dir, $epxresion_2$.type, expresion.type)

	getCode(expresion.dir '=' dir_1 '/' dir_2)}
expresion → $expresion_1$ % expresion	expresion → $expresion_1$ % expresion{ Si $expresion_1$.type = entero and $expresion_2$.type = entero Entonces expresion.type = max($expresion_1$.type, $expresion_2$.type) expresion.dir = newTemp() dir_1 = ampliar($expresion_1$.dir, $epxresion_1$.type, expresion.type) dir_2 = ampliar($expresion_2$.dir, $epxresion_2$.type, expresion.type) getCode(expresion.dir '=' dir_1 '+' dir_2) Sino Error("El módulo se aplica a enteros") Fin Si}
expresion →(expresion$_1$)	expresion →(expresion$_1$){expresion.type = $expresion_1$.type expresion.dir = $expresion_1$.dir}
expresion → variable	expresion → variable{expresion.type = variable.type expresion.dir = variable.dir}
expresion →num	expresion →num{expresion.type = num.type expresion.dir = num.dir}
expresion →cadena	expresion →cadena{ expresion.type ='string' Si TablaCadenas.existe(cadena) Entonces expresion.dir= TablaCadena.getIndexStr(cadena) Sino expresion.dir=TalbaCadena.insert(cadena) Fin Si}
expresion →caracter	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
	variable → id {
	Si STS.getTop().existe(id) Entonces
	IDGBL = id
	Sino
	Error(“No se ha declarado la variable”)
	Fin Si
	} variable comp {
	Si variable_com.code_est=true Entonces
	variable.dir=newTemp()
	variable.type = variable_com.type
	genCode(variable.dir ’=’ id’[’ variable_com.des’]’)
	variable.base = id.dir
	variable.code_est= true
	variable.des = variable_com.des
	Sino
	variable.dir = id)
	variable.type = STS.getTop().getType(id)
	variable.code_est= false

variable_comp → dato_est_sim	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_comp → arreglo	variable_comp → arreglo {variable_comp.type = arreglo.type variable_comp.des = arreglo.dir variable_comp.code_est = true}
variable_comp → (parametros)	variable_comp → (parametros) { Si STS.getGlobal().getVar(IDGBL)= ’func’ Entonces lista = STS.getGlobal().getListaArgs(IDGBL) num = STS.getGlobal().getNumArgs(IDGBL) Si num = parametros.num Entonces Para cada i = 0 hasta i = num hacer Si lista[i] != parametro.lista[i] entonces

	Error(“Los parámetros de la función no coinciden con la función”) Fin Si Fin Para Sino Error(“El número de parámetros no coincide con la función”) Fin Si}
dato_est_sim → dato_est_sim.id	dato_est_sim → dato_est_sim.id {Si dato_est_sim$_1$.estructura = true Entonces Si dato_est_sim$_1$.tabla.existe(id) Entonces dato_est_sim.des = dato_est_sim$_1$.des + dato_est_sim.tabla$_1$.getDir(id) typeTemp=dato_est_sim$_1$.tabla.getType(id) estTemp = dato_est_sim$_1$.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_sim$_1$.tabla.getType(id) FinSi dato_est_sim.code_est=true Sino Error(“La variable no ha sido declarada”) FinSi Sino Error(“La variable no es una estructura”) FinSi}
dato_est_sim →ε	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 → [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("El tamaño del arreglo no es un número entero") Fin Si Sino Error("La variable asociada no es tipo de arreglo") Fin Si}
arreglo → $arreglo_1$ [expresion]	arreglo → $arreglo_1$ [expresion]{arreglo.type = STS.getTop().getType($arreglo_1$.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'=' $arreglo_1$.dir '+' dirTemp) Sino Error("El tamaño del arreglo no es un número entero") Fin Si Sino Error("La variable asociada no es tipo de arreglo") Fin Si}

parametros → lista_param	parametros → lista_param{parametros.lista = lista_param.lista parametros.num = lista_param.num}
parametros→ε	parametros→ε {parametros.lista = NULO parametros.num = 0}
lista_param → $lista\_param_1$, expresion	lista_param → $lista\_param_1$, expresion{ lista_param.lista = $lista\_param_1$.lista lista_param.lista.append(expresion.type) lista_param.num = $lista\_param_1$ + 1}
lista_param → expresion	lista_param → expresion{ lista_param.lista = newList() lista_param.lista.append(expresion.type) lista_param.num = 1}