Universidad Nacional Autónoma de México
Compiladores
Definición Dirigida por Sintaxis para el lenguaje C2
Profesor: Adrián Ulises Mercado Martínez

PRODUCCIONES	**REGLAS SEMÁNTICAS**
programa → declaraciones	tstack.push() Gdir = 0 Gtype = -1 Gbase = -1
declaraciones → declaraciones$_1$ declaración	
declaraciones → ε	
declaración → tipo decl_fun_var	Gtype = tipo.type
declaración → decl_struct	
decl_fun_var → list_var **;**	
decl_fun_var → decl_fun	
list_var → list_var **,** var	
list_Var → var	
var → **id**	**Si** pila.top().TS.isIn(id) **Entonces** error(La variable ya fue declarada o no puede ser de tipo void) **Sino** pila.top.TS.addSymbo(id) pila.top.TS.setType(id, Gtype) pila.top.TS.setDir(id, Gdir) pila.top.TS.setTypeVar(id, "var") Gdir←Gdir + pila.top.TT.getTam(Gtype) **FinSi**
tipo → base comp_arr	Gbase = base.type tipo.type = comp_arr.type
tipo → **struct id**	**Si** pila.top().TS.isIn(id) **Entonces** **Si** pila.top().TS.getClas(id)= struct **Entonces**

PRODUCCIONES	REGLAS SEMÁNTICAS
	tipo.type = pila.top().getType(id) **Sino** error(El id no representa una estructura) **FinSi** **Sino Si** pila.global().TS.isIn(id) **Entonces** **Si** pila.global().TS.getClas(id)= struct **Entonces** tipo.type = pila.global().getType(id) **Sino** error(El id no representa una estructura) **FinSi** **FinSi**
comp_arr→ [numero] comp_arr$_1$	**Si** numero.val >0 y numero.type = int **Entonces** comp_arr.type = pila.top().TT.addType("array",num.val, comp_arr$_1$.type) **Sino** error(El indice debe ser mayor a cero y entero) **FinSi**
comp_arr→ ε	comp_arr.type = Gbase
base→**void**	base.type = void
base → **char**	base.type = char
base → **int**	base.type = int
base → **float**	base.type = float
base → **double**	base.type = double
decl_struct →**struct** {body_struct} list_var **;**	pila.push() pilaDir.push(dir) dir = 0 dir = pilaDir.pop() tabla = pila.pop() Gtype = pila.top().TT.addType('struct', tabla)
decl_struct → **struct id**{ body_struct } list_var **;**	pila.push() pilaDir.push(dir) dir = 0 dir = pilaDir.pop() tabla = pila.pop() Gtype = pila.top().TT.addType('struct', tabla) **Si** pila.top().TS.isIn(id) **Entonces**

PRODUCCIONES	REGLAS SEMÁNTICAS
	error(La variable ya fue declarada o no puede ser de tipo void) **Sino** pila.top.TS.addSymbo(id) pila.top.TS.setType(id, Gtype) pila.top.TS.setTypeVar(id, "struct") **FinSi**
body_struct → body_struct$_1$ decl_local	
body_struct → decl_local	
decl_fun→ **id (** lista_params **){** decl_locales bloqueSentencias **}**	GListaRetorno = nuevaLista() **Si** pila.global.existe(id) **Entonces** error(El id ya fue declarado) **Sino** **Desde** i =1 **hasta** GListaRetorno.num **hacer** **Si** GListaRetorno[i] $\neq$ Gtype **Entonces** error("La sentencia de retorno no devuelve el tipo correcto de dato") **FinSi** **FinDesde** pila.global().TS.addSymbo(id) pila.global().TS.setType(id, Gtype) pila.global().TS.setTypeVar(id, "func") pila.global().TS.setArgs(id, lista_params.lista) genCode("label",,,id) **FinSi**
list_params → params	list_params.lista = params.lista
list_params → ε	list_params.lista = nulo
params → params$_1$ **,** param	params$_1$.lista.add(param.type) params.lista = params$_1$.lista
params → param	params.lista = nuevaLista() params.lista.add(param.type)
param → type_param **id**	**Si** pila.top().TS.isIn(id) **Entonces** error(La variable ya fue declarada o no puede ser de tipo void) **Sino** pila.top.TS.addSymbo(id) pila.top.TS.setType(id, type_param.tyep) pila.top.TS.setTypeVar(id, "param")

PRODUCCIONES	REGLAS SEMÁNTICAS
	pila.top.TS.setDir(id, Gdir) Gdir←Gdir + pila.top.TT.getTam(type_param.type) **FinSi**
type_param → base parte_array	type = pila.top.TT.add("array", type_param.listaDim[1], base.type) **Desde** i = 2 **hasta** parte_array.listaDim.num **hacer** type = pila.top.TT.add("array", type_param.listaDim[1], type) **FinDesde** type_param.type =type
type_param → base	type_param.type = base.type
type_param → **struct id**	**Si** pila.top().TS.isIn(id) **Entonces** **Si** pila.top().TS.getClas(id)= struct **Entonces** type_param.type = pila.top().getType(id) **Sino** error(El id no representa una estructura) **FinSi** **Sino Si** pila.global().TS.isIn(id) **Entonces** **Si** pila.global().TS.getClas(id)= struct **Entonces** type_param.type = pila.global().getType(id) **Sino** error(El id no representa una estructura) **FinSi** **FinSi**
parte_array → parte_array$_1$ [**numero**]	parte_array$_1$.listaDim.add(num) parte_array.listaDim = parte_array1.listaDim
parte_array → [ ]	parte_array.listaDim = nuevaLista() parte_array.listaDim.add(-1)
decl_locales → decl_locales$_1$ decl_local	
decl_locales → ε	
decl_local → decl_var	
decl_local → decl_struct	
decl_var → tipo list_var **;**	Gtype = tipo.type
bloqueSentencias → sentencias	

PRODUCCIONES	REGLAS SEMÁNTICAS
bloqueSentencias $\rightarrow \varepsilon$	
sentencias $\rightarrow$ sentencias$_1$ sentencia	
sentencias $\rightarrow$ sentencia	
sentencia $\rightarrow$ sentIf	
sentencia $\rightarrow$ sentWhile	
sentencia $\rightarrow$ sentPuts	
sentencia $\rightarrow$ sentPutw	
sentencia $\rightarrow$ sentAsig	
sentencia $\rightarrow$ sentBreak	
sentencia $\rightarrow$ sentReturn	
sentencia $\rightarrow$ sentProc	
sentencia $\rightarrow$ sentFor	
sentencia $\rightarrow$ sentSwitch	
sentReturn $\rightarrow$ **return** expresion **;**	GlistaRetorno.add(expresion.type)
sentReturn $\rightarrow$ **return;**	GlistaRetorno.add(void)
sentProc $\rightarrow$ **id(** lista_args **);**	var = pila.global().getTypeVar(id) **Si** pila.global().existe(id) && var = "func" **Entonces** num=pila.global().getNumArgs(id) **Si** lista_args.num $\neq$ num **Entonces** error(El número de argumentos no coincide) **FinSi** lista = pila.global().getArgs(id) **Desde** i =1 **hasta** num **hacer** **Si** lista_args.lista[i] $\neq$ lista[i] **Entonces** error(El tipo del argumento i no es correcto) **FinSi** **FinDesde** GenCode("call",id, num,) **Sino** error(El id no existe o no hace

PRODUCCIONES	**REGLAS SEMÁNTICAS**
	referencia a una función) **FinSi**
lista_args → args	lista_arg.lista = args.lista
lista_args → ε	lista_args.lista = nulo
args → $args_1$, arg	$args_1$.lista.add(arg.type)
args → arg	args.lista = nuevaLista() args.lista.add(arg.type)
arg → epxresion	arg.type = expresion.type
expresion → $expresion_1$ + $expresion_2$	expresion = nuevaExpresion() expresion.tipo = max($expresion_1$.tipo, $expresion_2$.tipo) expresion.dir = nuevaTemp() α_1 = ampliar($expresion_1$.dir, $expresion_1$.type, expresion.type) α_2 = ampliar($expresion_2$.dir, $expresion_2$.type, expresion.type) genCode("+", α_1.dir, α_2.dir, expresion.dir)
expresion → $expresion_1$ - $expresion_2$	expresion = nuevaExpresion() expresion.tipo = max($expresion_1$.tipo, $expresion_2$.tipo) expresion.dir = nuevaTemp() α_1 = ampliar($expresion_1$.dir, $expresion_1$.type, expresion.type) α_2 = ampliar($expresion_2$.dir, $expresion_2$.type, expresion.type) genCode("−", α_1.dir, α_2.dir, expresion.dir)
expresion → $expresion_1$ * $expresion_2$	expresion = nuevaExpresion() expresion.tipo = max($expresion_1$.tipo, $expresion_2$.tipo) expresion.dir = nuevaTemp() α_1 = ampliar($expresion_1$.dir, $expresion_1$.type, expresion.type) α_2 = ampliar($expresion_2$.dir, $expresion_2$.type, expresion.type) genCode("∗", α_1.dir, α_2.dir, expresion.dir)
expresion → $expresion_1$ / $expresion_2$	expresion = nuevaExpresion() expresion.tipo =

PRODUCCIONES	REGLAS SEMÁNTICAS
	max(expresion$_1$.tipo, expresion$_2$.tipo) expresion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, expresion.type) α_2 = ampliar(expresion$_2$.dir, expresion$_2$.type, expresion.type) genCode("/", α_1.dir, α_2.dir, expresion.dir)
expresion $\rightarrow$ (expresion$_1$)	expresion = expresion$_1$
expresion $\rightarrow$ expresion$_1$ % expresion$_2$	expresion = nuevaExpresion() **Si** expresion$_1$.tipo = expresion$_2$.tipo = int **entonces** expresion.tipo = expresion$_1$.tipo expresion.dir = nuevaTemp() genCode("%", expresion$_1$.dir, expresion$_2$.dir, expresion.dir) **Sino** error(El tipo del operando debe ser entero) **FinSi**
expresion $\rightarrow$ **id** complemento	Gid = id expresion.tipo = complemento.type expresion.dir = complento.dir
expresion $\rightarrow$ **numero**	expresion.tipo = numero.type expresion.dir = numero
expresion $\rightarrow$ **caracter**	expresion.tipo = char expresion.dir = caracter
complemento $\rightarrow$ comp_struct	**Si** comp_struct.base = Gid **Entonces** complemento.dir = Gid complemento.type = comp_struct.type **Sino** complemento.dir = Gid "[" comp_struct.des"]" complemento.type = comp_struct.type **FinSi**
complemento $\rightarrow$ array	complemento.dir = Gid "[" array.dir"]" complemento.type = array.type
complemento $\rightarrow$ **(** list_param **);**	var = pila.global().getTypeVar(Gid) **Si** pila.global().existe(Gid) && var = "func"**Entonces** num=pila.global().getNumArgs(Gid) **Si** lista_args.num $\neq$ num **Entonces** error(El número de argumentos no coincide)

PRODUCCIONES	REGLAS SEMÁNTICAS
	FinSi lista = pila.global().getArgs(Gid) **Desde** i =1 **hasta** num **hacer** **Si** lista_args.lista[i] $\neq$ lista[i] **Entonces** error(El tipo del argumento i no es correcto) **FinSi** **FinDesde** complemento.dir = nuevaTemporal() complemento.type = pila.global().TS.getType(Gid) GenCode("call",id, num,) **Sino** error(El id no existe o no hace referencia a una función) **FinSi**
array $\rightarrow$ array$_1$ [expresion]	tipo = pila.top.TT.getName(array1.tipo) **Si** tipo = array **Entonces** **Si** expresion.tipo = int **Entonces** temp = nuevaTemporal() array.dir = nuevaTemporal() array.tipo = pila.top.TT.getTipoBase(tipo) array.tam = pila.top.TT.getTam(array.tipo) genCode("$*$", expresion.dir, array.tam, temp) genCode("$+$", array$_1$.dir, temp, array.dir) **Sino** error(El índice para un arreglo debe ser entero) **FinSi** **Sino** error(El arreglo no tiene mas dimensiones) **FinSi**
array $\rightarrow$ [expresion]	**Si** pila.top.TS.isIn(Gid) **Entonces** tipo = pila.top.TS.getType(Gid) **Si** pila.top.TT.getName(tipo) = array **Entonces** **Si** expresion.tipo = int **Entonces** array.dir = nuevaTemporal() array.tipo = pila.top.TT.getTipoBase(tipo) array.tam = pila.top.TT.getTam(array.tipo) genCode("$*$", expresion.dir, array.tam, array.dir) **Sino** error(El indice debe ser entero)

PRODUCCIONES	REGLAS SEMÁNTICAS
	FinSi **Sino** error(El id no es un array) **FinSi** **Sino Si** pila.top.TS.isIn(Gid) **Entonces** tipo = pila.top.TS.getType(Gid) **Si** pila.top.TT.getName(tipo) = array **Entonces** **Si** expresion.tipo = int **Entonces** array.dir = nuevaTemporal() array.tipo = pila.top.TT.getTipoBase(tipo) array.tam = pila.top.TT.getTam(array.tipo) genCode(" $*$ ", expresion.dir, array.tam, array.dir) **Sino** error(El indice debe ser entero) **FinSi** **Sino** error(El id no es un array) **FinSi** **Sino** error(El id no fue declarado) **FinSi**
condicion $\rightarrow$ condicion$_1$ $\|\|$ condicion$_2$	codicion = nuevaExpresion() condicion.tipo = max(condicion$_1$.tipo, condicion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(condicion$_1$.dir, condicion$_1$.type, condicion.type) α_2 = ampliar(condicion$_2$.dir, condicion$_2$.type, condicion.type) genCode("$\|\|$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ condicion$_1$ && condicion$_2$	codicion = nuevaExpresion() condicion.tipo = max(condicion$_1$.tipo, condicion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(condicion$_1$.dir, condicion$_1$.type, condicion.type) α_2 = ampliar(condicion$_2$.dir, condicion$_2$.type, condicion.type) genCode("$\&\&$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ expresion$_1$ $==$ expresion$_2$	condicion = nuevaExpresion()

PRODUCCIONES	**REGLAS SEMÁNTICAS**
	expresion.tipo = max(expresion$_1$.tipo, expresion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, condicion.type) α_2 = ampliar(expresion$_2$.dir, expresion$_2$.type, condicion.type) genCode("$==$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ expresion$_1$! $=$ expresion$_2$	condicion = nuevaExpresion() expresion.tipo = max(expresion$_1$.tipo, expresion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, condicion.type) α_2 = ampliar(expresion$_2$.dir, expresion$_2$.type, condicion.type) genCode("! $=$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ expresion$_1$ $<$ expresion$_2$	condicion = nuevaExpresion() expresion.tipo = max(expresion$_1$.tipo, expresion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, condicion.type) α_2 = ampliar(expresion$_2$.dir, expresion$_2$.type, condicion.type) genCode("$<$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ expresion$_1$ $>$ expresion$_2$	condicion = nuevaExpresion() expresion.tipo = max(expresion$_1$.tipo, expresion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, condicion.type) α_2 = ampliar(expresion$_2$.dir, expresion$_2$.type, condicion.type) genCode("$>$", α_1.dir, α_2.dir, condicion.dir)
condicion $\rightarrow$ expresion$_1$ $>=$ expresion$_2$	condicion = nuevaExpresion() expresion.tipo = max(expresion$_1$.tipo, expresion$_2$.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion$_1$.dir, expresion$_1$.type, condicion.type) α_2 = ampliar(expresion$_2$.dir,

PRODUCCIONES	**REGLAS SEMÁNTICAS**
	expresion_2.type, condicion.type) genCode(">=", α_1.dir, α_2.dir, condicion.dir)
condicion → expresion_1 <= expresion_2	condicion = nuevaExpresion() expresion.tipo = max(expresion_1.tipo, expresion_2.tipo) condicion.dir = nuevaTemp() α_1 = ampliar(expresion_1.dir, expresion_1.type, condicion.type) α_2 = ampliar(expresion_2.dir, expresion_2.type, condicion.type) genCode("<=", α_1.dir, α_2.dir, condicion.dir)
condicion → **!** condicion_1	condicion.dir = nuevaTemporal() condicion.type = condicion_1.type genCode("!", condicion_1.dir, " , condicion.dir)
sentIf → **if**(condicion) bloqueOSentencia sentElse	numLabel = numLabel +1 pilaLabel.push(numLabel) genCode('ifFalse', condicion.dir, 'goto', 'LELSE'+pilaLabel.cima) genCode('goto', " , " , 'LFIN+pilaLabel.cima) genCode('label', " , " , 'LELSE'+pilaLabel.cima) genCode('label', " , " , 'LFIN'+pilaLabel.cima) pilaLabel.pop()
sentElse → **else** bloqueOSentencia	
sentElse → ε	
sentWhile → **while**(condicion) bloqueOSentencia	numLabel = numLabel +1 pilaLabel.push(numLabel) genCode('label', " , " , 'LINI'+pilaLabel.cima) genCode('ifFalse', condicion.dir, 'goto', 'LFIN'+pilaLabel.cima) genCode('goto', " , " , 'LINI+pilaLabel.cima) genCode('label', " , " , 'LFIN'+pilaLabel.cima) pilaLabel.pop()
sentSwitch → **switch(**expresion**)**{body_switch}	numLabel = numLabel + 1 sLabelSwitch = nuevaPila() numLabelSwitch = -1 sDirSwitch = nuevaPila() sLabel.push(numLabel) Validar que expresión sea de tipo entero genCode('goto', " , " , 'LINI'+pilaLabel.cima)

PRODUCCIONES	REGLAS SEMÁNTICAS
	genCode('goto', '', '', 'LFIN'+pilaLabel.cima) genCode('label', '', '', 'LINI'+pilaLabel.cima) **Si** sDirSwitch.top = 'default' **Entonces** temporal = sLabelSwitch.pop() sDirSwitch.pop() existe = verdadero **FinSi** **Mientras** sLabelSiwtch.top $\neq$ nulo **Hacer** t = nuevaTemporal() genCode('==', expresion.dir, sDirSwitch.pop(), t) genCode('if', t, 'goto', 'LIC'+sLabelSwitch.pop()) **FinMientras** **Si** existe **Entonces** genCode('goto', '', '', 'LIC'+temporal) **FinSi** genCode('label', '', '', 'LFIN'+pilaLabel.cima) sLabel.pop()
body_switch $\rightarrow$ caso body_switch	
body_switch $\rightarrow$ predeterminado	
body_switch $\rightarrow \varepsilon$	
caso $\rightarrow$**case** expresion **:** sentencias	Validar que expresión sea tipo entero numLabelSwitch = numLabelSwitch + 1 sLabelSwitch.push(numLabelSwitch) genCode('label', '', '', 'LIC'+sLabelSwitch.cima) sDirSWitch.push(expresion.dir)
predeterminado $\rightarrow$ **defualt:** sentencias	numLabelSwitch = numLabelSwitch + 1 sLabelSwitch.push(numLabelSwitch) genCode('label', '', '', 'LIC'+sLabelSwitch.cima) sDirSWitch.push(defualt)
sentFor $\rightarrow$ **for**(sentAsig$_1$**;** condicion**;** sentAsig$_2$) bloqueOSentencia	numLabel = numLabel + 1 sLabel.push(numLabel) genCode('label', '', '', 'LINI'+pilaLabel.cima) genCode('ifFalse', condicion.dir, 'goto', 'LFIN'+pilaLabel.cima) genCode(sentAsig$_2$) genCode('goto', '', '', 'LINI+pilaLabel.cima) genCode('label', '', '', 'LFIN'+pilaLabel.cima) sLabel.pop()
sentAsig $\rightarrow$ left_part **=** expresion;	t = reducir(expresion.dir,

PRODUCCIONES	REGLAS SEMÁNTICAS
	expresion.tipo, left_part.type) genCode('=', t, '', left_part.dir)
left_part → **id** comp_struct	Gid = id **Si** comp_struct.base = Gid **Entonces** left_part.dir = Gid left_part.type = comp_struct.type **Sino** left_part.dir = Gid "[" comp_struct.des"]" left_part.type = comp_struct.type **FinSi**
left_part → **id** array	Gid = id complemento.dir = Gid "[" array.dir"]" complemento.type = array.type
comp_struct → comp_struct$_1$ **. id**	**Si** comp_struct1.clase = struct **Entonces** **Si** comp_struct$_1$.tabla.isIn(id) **Entonces** comp_struct.des = comp_struct$_1$.des + comp_struct.tabla.TS.getDir(id) comp_struct.clase = comp_struct$_1$.TS.getClase(id) comp_struct.type = comp_struct$_1$.TS.getType(id) comp_struct.tabla = comp_struc$_1$.TT.getBase() comp_struct.base = id **Sino** error(El id no es un miembro de la estructura) **FinSi** **Sino** error(El comp_struct$_1$.base no es una estrcutrua) **FinSi**
comp_struct → ε	**Si** pila.top().TS.isIn(Gid) **Entonces** comp_struct.clase = pila.top().TS.getClase(Gid) comp_struct.type = pila.top().TS.getType(Gid) comp_struct.tabla = pila.top().TT.getBase() comp_struct.base = Gid comp_struct.des = 0 **Sino Si** pila.global().TS.isIn(Gid) **Entonces** comp_struct.clase = pila.global().TS.getClase(Gid) comp_struct.type = pila.global().TS.getType(Gid) comp_struct.tabla = pila.global().TT.getBase() comp_struct.des = 0 comp_struct.base = Gid

PRODUCCIONES	REGLAS SEMÁNTICAS
	Sino error(El id no fue declarado) **FinSi**
sentPutw → **print(** expresion **);**	
sentPuts → **print(cadena);**	
sentBreak → **break;**	genCode('goto', '', '' , 'LFIN'+sLabel.cima())
bloqueOSentencia → { bloqueSentencias } bloqueOSentencia → sentencia	