Definición dirigida por sintaxis

Concha Vázquez Miguel
mconcha@ciencias.unam.mx

Flores Martínez Andrés
andresfm97@ciencias.unam.mx

Gladín García Ángel Iván
angelgladin@ciencias.unam.mx

Sánchez Pérez Pedro Juan Salvador
pedro_merolito@ciencias.unam.mx

Vázquez Salcedo Eduardo Eder
eder.vs@ciencias.unam.mx

14 de diciembre de 2018

Reglas gramaticales	Reglas semánticas
prog → decls funcs	
$decls$ → tipo lista SEMICOLON decls_1	$decls$.cantidad = lista.cantidad + $decls_1$.cantidad
decls → ϵ	decls.cantidad = 0
tipo→ INTTYPE	tipo.type = INT tipo.bytes = 4
tipo → FLOATTYPE	tipo.type = FLOAT tipo.bytes = 4
tipo → DOUBLETYPE	tipo.type = DOUBLE tipo.bytes = 8
tipo → CHARTYPE	tipo.type = CHAR tipo.bytes = 1
tipo → VOID	tipo.type = VOID tipo.bytes = 1
tipo → STRUCT LCURLYB decls RCURLYB	tipo.bytes = decls.cantidad tipo.symbol_table = new_symbol_table(decls) tipo.type = global_table.size -1
$lista$ → $lista_1$ COMMA ID arreglo	$lista$.cantidad = $lista_1$.cantidad + arreglo.bytes insert_symbol(id=ID,type=arreglo.type,dir=new Dir())
$lista$ → ID arreglo	$lista_1$.cantidad = arreglo.bytes insert_symbol(id=ID,type=arreglo.type,dir=new Dir())
numero→ signo INT	numero.type = INT.yylval.type numero.val = signo*INT.yylval.val
numero → signo DOUBLE	numero.type = DOUBLE.yylval.type numero.val = signo* DOUBLE.yylval.val
numero → signo FLOAT	numero.type = FLOAT.yylval.type numero.val = signo* FLOAT.yylval.val
signo→ PLUS	signo = 1
signo → MINUS	signo = -1
signo → ϵ	signo = 1
$arreglo$ → LBRACKET numero RBRACKET $arreglo_1$	$arreglo$.tamaño = $arreglo_2$.tamaño+1 $arreglo$.dims = $arreglo$.dims.append(numero.val) insert_type(type=array,tam=numero.val,base=$arreglo_1.type$)
arreglo → ϵ	arreglo.tamaño = 0

$funcs \rightarrow$ FUNC tipo ID LPAR args RPAR LCURLYB decls sents RCURLYB $funcs_1$	funcs.code = funcs.code || newLabel(): ID insert_symbol(id=ID,type=tipo.type,dir=newDir(),.. ..var=1,args=args.lista_args) insert_type(type=func,tam=decls.cantidad,tabla=tope_tablas())
funcs $\rightarrow \epsilon$	
args$\rightarrow$ lista_args	args.num = $lista_a rgs$.num args.$lista_a rgs$ = $lista_a rgs.lista_a rgs$
args $\rightarrow \epsilon$	args.num = 0
$lista\_args \rightarrow lista\_args_1$ COMMA tipo ID parte_arr	$lista\_args$.num = $lista\_arg_1$.num + 1 $lista\_args$.lista_args = $lista\_args_1$.lista_args $lista\_args.append(type = tipo.type, parte\_arr)$ insert_type(type=tipo.type,tam=tipo.bytes,.. ..,base=tipo.type) insert_symbol(id=ID,tipo=tipo.type,dir=newDir())
lista_args $\rightarrow$ tipo ID parte_arr	$lista\_args$.num = 1 $lista\_args.append(type = tipo.type, parte\_arr)$ insert_type(tipo=tipo.type,tam=tipo.tipo.bytes,base=tipo.type) insert_symbol(id=ID,tipo=tipo.type,dir=newDir())
$parte\_arr \rightarrow$ LBRACKET RBRACKET $parte\_arr_1$	$parte\_arr = parte\_arr_1$+1
parte_arr $\rightarrow \epsilon$	$parte\_arr = 0$
$sents \rightarrow$ sents1 sent	sents.code = sents.code ||.. ..get_first(sents)::back_patch(sents,newLabel())
$sent \rightarrow$ IF LPAR cond RPAR $sent_1$ sentp	sent.code = get_first(cond.trues) sent.code.concat(if cond.dir goto get_first(sent2)) sent.ifelse = merge(cond.falses, sentp.siguientes) sent.not_ifelse = merge(sent2, sentp.siguientes)
$sent \rightarrow$ WHILE LPAR cond RPAR $sent_1$	$sent$.siguientes = $cond$.falses sent.code = if cond.dir goto __ backpatch(cond.trues, newLabel()) backpatch(cond.falses, newLabel())
$sent \rightarrow$ DO $sent_1$ WHILE LPAR cond RPAR SEMICOLON	sent.code = sent2.code $sent$.siguientes = $cond$.falses concat(sent.code, if cond.dir goto __) backpatch(cond.trues, newLabel()) backpatch(cond.falses, newLabel())
$sent \rightarrow$ FOR LPAR $assign_1$ SEMICOLON cond SEMICOLON $assign_2$ RPAR $sent_2$	sent.code = assign1.code $sent$.siguientes = $cond$.falses concat(sent.code, if cond.dir goto __) backpatch(cond.trues, newLabel()) backpatch(cond.falses, newLabel()) concat(sent1.code, assign2.code)
sent$\rightarrow$ assign SEMICOLON	concat(sent.code, assign.arr_codigo, assign.count_codigo)
sent$\rightarrow$ RETURN exp SEMICOLON	sent.code = goto to_determine(newLabel())
sent$\rightarrow$ RETURN SEMICOLON	sent.code = goto to_determine(newLabel())
sent$\rightarrow$ LCURLYB sents RCURLYB	sent = sents back_patch(sents, newLabel())
sent$\rightarrow$ SWITCH LPAR exp RPAR LCURLYB casos RCURLYB	sent.code = casos.code
sent$\rightarrow$ BREAK SEMICOLON	sent.code = goto to_determine(newLabel())
sent$\rightarrow$ PRINT exp SEMICOLON	sent = create_list(newIndex())

assign→ parte_izq ASSIG exp SEMICOLON	assign.count_codigo =.. .. f1(parte_izq.id1, parte_izq.id2, exp, parte_izq.type) assign.arr_codigo =.. .. f2(parte_izq.id1, parte_izq.id2, exp, parte_izq.type)
sentp→ ϵ IFX	sentp.ifelse = false
sentp→ ELSE sent	sentp.ifelse = true sentp.siguientes = sent.siguientes assign.code = pop_label(lfalses)::
$casos \rightarrow$ CASE numero sent $casos_1$	casos.trues = create_list(newIndex()) casos.falses = create_list(newIndex()) t = newTemp() casos.code = t = tope_dir == numero.val concat(casos.code, if t goto newLabel()) concat(casos.code, goto casos1.siguiente) push_label(lfalses, get_first(casos.falses)
casos → DEFAULT sent	casos.code = sent.code
casos → ϵ	
parte_izq→ ID	parte_izquierda.id1 = ID
parte_izq → var_arr	parte_izquierda.id1 = var_arr.representacion parte_izquireda.type = var_arr.type
parte_izq → ID_1 DOT ID_2	parte_izquierda.id1 = ID_1 parte_izquierda.id2 = ID_2
var_arr→ ID LBRACKET exp RBRACKET	*var_arr*.representacion = representa(ID, exp) var_arr.type = get_type(pila_tablas, ID) var_arr.basico = get_base(pila_tablas, var_arr.type) var_arr.indice_tamanios = var_arr.indice_tamanios + 1 var_arr.type_table = find_table(ID)
$var\_arr \rightarrow var\_arr_1$ LBRACKET exp RBRACKET	var_arr.type = get_type(pila_tablas, ID) var_arr.basico = get_base(pila_tablas, var_arr.type) var_arr.indice_tamanios = var_arr.indice_tamanios + 1 var_arr.type_table = var_arr1.type_table
$exp \rightarrow exp_1$ PLUS exp_2	exp1 = suma(exp1, exp2)
$exp \rightarrow exp_1$ MINUS exp_2	exp1 = resta(exp1, exp2)
$exp \rightarrow exp_1$ PROD exp_2	exp1 = multiplicacion(exp1, exp2)
$exp \rightarrow exp_1$ DIV exp_2	exp1 = division(exp1, exp2)
$exp \rightarrow exp_1$ MOD exp_2	exp1 = modulo(exp1, exp2)
exp → var_arr	exp = envolver_varr(var_arr) exp.tipo_basico = var_arr.tipo_basico exp.dims = var_arr.dims
exp → ID	exp = identificador(ID)
exp → CADENA	exp = envolver_cadena(CADENA)
exp → numero	exp = get_numero(numero)
exp → CARACTER	exp = envolver_caracter(CARACTER)
exp → ID LPAR params RPAR	t = newTemp() exp.code = (t = call ID params.count)
params→ lista_param	params.p = lista_param.p params.count = lista_param.count params.lista_tipos = lista_param.lista_tipos
params → ϵ	params.p = 0 params.count = 0

$lista\_param \rightarrow lista\_param_1$ COMMA exp	lista_param.p = lista_param1.p + 1 lista_param.count = lista_param1.count + 1 insert(lista_param.lista_tipos.tipo_basico, exp.tipo_basico) insert(lista_param.lista_tipos.dims, exp.dims)
lista_param → exp	lista_param.p = 1 lista_param.count = 1 insert(lista_param.lista_tipos.tipo_basico, exp.tipo_basico) insert(lista_param.lista_tipos.dims, exp.dims)
$cond \rightarrow cond_1$ OR $cond_2$	cond.trues = merge(cond1.trues, cond2.trues) cond.falses = cond2.falses backpatch(cond2.falses, newLabel())
$cond \rightarrow cond_1$ AND $cond_2$	cond.trues = cond2.trues cond.falses = merge(cond1.falses, cond2.falses) backpatch(cond1.trues, newLabel())
$cond \rightarrow$ NOT $cond_1$	cond.trues = cond1.falses cond.falses = cond1.trues
$cond \rightarrow$ LPAR $cond_1$ RPAR	cond.trues = cond1.trues cond.falses = cond1.falses
cond → exp_1 rel exp_2	cond.trues = create_list(newLabel()) cond.falses = create_list(newLabel()) cond.code = t = exp1.dir rel.val exp2.dir concat(cond.code, IF t GOTO newLabel()) concat(cond.code, GOTO newLabel())
cond → TRUE	cond.trues = create_list(newLabel()) cond.code = GOTO newLabel()
cond → FALSE	cond.falses = create_list(newLabel()) cond.code = GOTO newLabel()
rel→ LT	rel.tipo = LESS_THAN
rel → GT	rel.tipo = GREATER_THAN
rel → LEQ	rel.tipo = LESS_OR_EQUAL_THAN
rel → GEQ	rel.tipo = GREATER_OR_EQUAL_THAN
rel → NEQ	rel.tipo = NOT_EQUAL
rel → EQ	rel.tipo = EQUALS