Definición Dirigida por Sintaxis

Compiladores

Equipo:

Semestre 2020-2

Aquino Santiago Gerardo Rogelio
Moreno Osuna Isaac
Rubio Carmona Alonso Rafael
Carlos H.

Producción	Reglas Semánticas
programa → declaraciones funciones	STS. push(nuevaTS( )) STS. push(nuevaTS( )) dir = 0 programa. code = funciones. code
declaraciones → tipo lista_var ; declaraciones	Tipo = tipo. tipo
declaraciones → tipo_registro lista_var ; declaraciones	Tipo = tipo_registro. tipo
declaraciones → ε	declaraciones. tipo = base
tipo_registro → estructura inicio declaraciones fin	STS. push(nuevaTS( )) STT. push(nuevaTT( )) Sdir. push(dir) dir = 0 dir = Sdir. pop( ) Ts = STS. pop( ) Tt = STT. pop( ) ts. tt = Tt T. tipo = STT. getCima( ). append('struct', tam, Ts)
tipo → base tipo_arreglo	Base = base. base Tipo. tipo = tipo_arreglo. tipo
base → ent	base. base = ent
base → real	base. base = real
base → dreal	base. base = dreal
base → car	base. base = car
base → sin	base. base = sin
tipo_arreglo → (num) tipo_arreglo_1	Si num. tipo = ent Entonces Si num. dir > 0 Entonces tipo_arreglo. tipo = TT. append("arreglo", num. dir, tipo_arreglo_1. tipo) Sino Error("El valor debe ser positivo") Fin Si Sino Error("El valor debe ser entero") Fin Si
tipo_arreglo → ε	tipo_arreglo = base
lista_var → lista_var_1, id	Si ! Ts. existe(id) Entonces STS. getCima( ). append(id, dir, Tipo, 'var', nulo, −1) Dir ← dir + STT. getCima( ). getTam(Tipo) Sino Error("El id ya fue declarado anteriormente")

	Fin Si
lista_var → id	Si ! Ts. existe(id) Entonces STS. getCima( ). append(id, dir, Tipo, 'var', nulo, −1) Dir ← dir + STT. getCima( ). getTam(Tipo) Sino Error("El id ya fue declarado anteriormente") Fin Si
funciones → def tipo id(argumentos)inicio declaraciones sentencias fin funciones	Si ! STS. getCima( ). existe(id) Entonces STS. push(nuevaTS( )) Sdir. push(dir) dir = 0 lista_retorno = nuevaLista( ) Si cmpRet(lista_retorno, T. tipo) Entonces L = nuevaEtiqueta( ) backpatch(S. nextlist, L) F. code = etiqueta(id) || S. code || etiqueta(L) Sino Error("El valor no corresponde al tipo de la funcion") Fin Si STS. pop( ) dir = Sdir. pop() Sino Error("El id ya fue declarado") Fin Si
funciones → ε	funciones. dir = null
argumentos → lista_arg	argumentos. lista = lista_arg. lista argumentos. num = lista_arg. num
argumentos → sin	argumentos. lista = nulo argumentos. num = 0
lista_arg → lista_arg$_1$, arg	lista_arg. lista = lista_arg$_1$. lista lista_arg. lista. append(arg. Tipo) lista_arg. num = lista_arg. num + 1
lista_arg → arg	lista_arg. lista = lista_arg$_1$. lista lista_arg. lista. append(arg. Tipo) lista_arg. num = lista_arg. num + 1
arg → tipo_arg id	arg. tipo = tipo_arg. tipo
tipo_arg → base param_arr	Base = base. tipo tipo_arg. tipo = param_arr. tipo
param_arr → ( )param_arr$_1$	param_arr. tipo = STT. append("array", −, param_arr$_1$. tipo)
param_arr → ε	param_arr. tipo = Base
sentencias → sentencias$_1$ sentencia	L = nuevaEtiqueta( ) backpatch(sentencias$_1$. nextlist, L) sentencias. nextlist = sentencia. nextlist sentencias. code = sentencias$_1$. code || etiqueta(L) || sentencia. code
sentencias → sentencia	sentencias. nextlist = sentencia. nextlist sentencias. code = sentencia. code
sentencia → si e_bool entonces sentencia$_1$ fin	L = nuevaEriqueta( ) backpatch(e_bool. truelist, L)

	sentencia. nextlist = combinar(e_bool. falselist, sentencia$_1$. nextlist) sentencia. code = e_bool. code || etiqueta(L) || sentencia$_1$. code
sentencia **→ si e_bool entonces sentencia$_1$ sino sentencia$_2$ fin**	L_1 = nuevaEtiqueta( ) L_2 = nuevaEtiqueta( ) backpatch(e_bool. truelist, L_1) backpatch(e_bool. falselist, L_2) sentencia. nextlist = combinar(sentencia$_1$. nextlist, sentencia$_2$. nextlist) sentencia. code = e_{bool}. code ||etiqueta(L_1)||sentencia$_1$. code || gen('goto'sentencia$_1$. nextlist[0]) ||etiqueta(L_2) || sentencia$_2$. code
sentencia → mientras e_bool hacer sentencia$_1$ fin	L_1 = nuevaEtiqueta( ) L_2 = nuevaEtiqueta( ) backpatch(sentencia$_1$. nextlist, L_1) backpatch(e_bool. truelist, L_2) sentencia. nextlist = e_bool. falselist sentencia. code = etiqueta(L_1) || e_bool. code || etiqueta(L_2) || sentencia$_1$. code || gen('goto'sentencia$_1$. nextlist[0])
sentencia → hacer sentencia$_1$ mientras e_bool;	L_1 = nuevaEtiqueta( ) L_2 = nuevaEtiqueta( ) backpatch(sentencia$_1$. nextlist, L_1) backpatch(e_bool. truelist, L_2) sentencia. nextlist = e_bool. falselist sentencia. code = etiqueta(L_2) || sentencia$_1$. code || etiqueta(L_1) || e_bool. code || gen('goto'sentencia$_1$. nextlist[0])
sentencia **→ segun (variable)hacer casos predeterminado fin**	L_1 = nuevaEtiqueta( ) L_2 = nuevaEtiqueta( ) backpatch(sentencia. truelist, L_1) backpatch(sentencia. falselist, L_2) sentencia. nextlist = combinar(casos. nextlist, predeterminado. nextlist) sentencia. code = variable. code || etiqueta(L_1) || casos. code || gen('goto' casos. nextlist[0]) || etiqueta(L_2) || predeterminado. code
sentencia → variable := expresion;	sentencia. nextlist = null Si TS. existe(variable) Entonces tipo_variable = TS. getTipo(variable) t = reducir(expresion. dir, expresion. tipo, tipo_variable) sentencia. codegen = variable(' =' t) Sino error(La variable no ha sido declarada) Fin Si
sentencia → escribir expresion;	sentencia. code = gen("printf" expresion. dir) sentencia. nextlist = null
sentencia → leer variable;	sentencia. code = gen("scanf" variable. dir) sentencia. nextlist = null
sentencia → devolver;	sentencia. code = gen("return")

	sentencia. nextlist = null
sentencia → devolver expresion;	lista_retorno. append(expresion. tipo) sentencia. code = gen(returnexpresion. dir) sentencia. nextlist = null
sentencia → terminar;	L = nuevaEtiqueta( ) sentencia. code = gen('goto' L) sentencia. nextlist = nuevaLista( ) sentencia. nextlist. add(L)
sentencia → inicio sentencias fin	sentencia. nextlist = sentencias. nextlist
casos → caso num: sentencia casos$_1$	L_1 = nuevaEtiqueta( ) L_2 = nuevaEtiqueta( ) backpatch(num. truelist, L_1) backpatch(num. falselist, L_2) casos. nextlist = combiar(sentencia. nextlist, casos$_1$. nextlist) casos. code = num. dir || etiqueta(L_1) || sentencia. code || gen('goto' sentencia. nextlist[0]) || etiqueta(L_2) || casos$_1$. code
casos → caso num: sentencia	L_1 = nuevaEtiqueta( ) backpatch(num. truelist, L_1) casos. code = num. dir || etiqueta(L_1) || sentencia. code || gen('goto' sentencia. nextlist[0])
predeterminado → pred: sentencia	L_1 = nuevaEtiqueta( ) backpatch(num. falselist, L_1) predeterminado. code = num. dir || etiqueta(L_1) || sentencia. code || gen('goto' sentencia. nextlist[0])
predeterminado → ε	predeterminado. dir = null
e_bool → e_bool$_1$ o e_bool$_2$	L = nuevaEtiqueta( ) backpatch(e_bool$_1$. flaselist, L) e_bool. truelist = combinar(e_bool$_1$. truelist, e_bool$_2$. truelist) e_bool. falselist = e_bool$_2$. falselist e_bool. code = e_bool$_1$. code || etiqueta(L) || e_bool$_2$. code
e_bool → e_bool$_1$ y e_bool$_2$	L = nuevaEtiqueta( ) backpatch(e_bool$_1$. truelist, L) e_bool. truelist = e_bool$_2$. truelist e_bool. falselist = combinar(e_bool$_1$. falselist, e_bool$_2$. falselist) e_bool. code = e_bool$_1$. code || etiqueta(L) || e_bool$_2$. code
e_bool → no e_bool$_1$	e_bool. truelist = e_bool$_1$. falselist e_bool. falselist = e_bool$_1$. truelist e_bool. code = e_bool$_1$. code
e_bool → relacional	e_bool. truelist = relacional. truelist e_bool. falselist = relacional. falselist
e_bool → verdadero	t_0 = nuevoIndice( ) e_bool. truelist = crearLista(t_0) e_bool. code = gen('goto' t_0)
e_bool → falso	t_0 = nuevoIndice( ) e_bool. falselist = crearLista(t_0) e_bool. code = gen('goto' t_0)
relacional → relacional$_1$ > relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo)

	relacional. code = gen(relacional. dir = $t_1' >' t_2$)
relacional → relacional$_1$ < relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo) relacional. code = gen(relacional. dir = $t_1' <' t_2$)
relacional → relacional$_1$ <= relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo) relacional. code = gen(relacional. dir = $t_1' <=' t_2$)
relacional → relacional$_1$ >= relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo) relacional. code = gen(relacional. dir = $t_1' >=' t_2$)
relacional → relacional$_1$ <> relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo) relacional. code = gen(relacional. dir = $t_1' <>' t_2$)
relacional → relacional$_1$ = relacional$_2$	relacional. dir = nuevaTemporal relacional. tipo = max (relacional$_1$. tipo, relacional$_2$. tipo) t_1 = ampliar(relacional$_1$. dir, relacional$_1$. tipo, relacional. tipo) t_2 = ampliar(relacional$_2$. dir, relacional$_2$. tipo, relacional. tipo) relacional. code = gen(relacional. dir = $t_1' =' t_2$)
relacional → expresion	relacional. dir = expresion. dir relacional. code = expresion. code
expresion → expresion$_1$ + expresion$_2$	expresion. dir = nuevaTemporal expresion. tipo = max(expresion$_1$. tipo, expresion$_2$. tipo) t_1 = ampliar(expresion$_1$. dir, expresion$_1$. tipo, expresion. tipo) t_2 = ampliar(expresion$_2$. dir, expresion$_2$. tipo, expresion. tipo) expresion. code = gen(expresion. dir$' =' t_1'+' t_2$)
expresion → expresion$_1$ − expresion$_2$	expresion. dir = nuevaTemporal expresion. tipo = max(expresion$_1$. tipo, expresion$_2$. tipo) t_1 = ampliar(expresion$_1$. dir, expresion$_1$. tipo, expresion. tipo) t_2 = ampliar(expresion$_2$. dir, expresion$_2$. tipo, expresion. tipo) expresion. code = gen(expresion. dir$' =' t_1'-' t_2$)
expresion → expresion$_1$ * expresion$_2$	expresion. dir = nuevaTemporal expresion. tipo = max(expresion$_1$. tipo, expresion$_2$. tipo) t_1 = ampliar(expresion$_1$. dir, expresion$_1$. tipo, expresion. tipo) t_2 = ampliar(expresion$_2$. dir, expresion$_2$. tipo, expresion. tipo) expresion. code = gen(expresion. dir$' =' t_1' *' t_2$)
expresion → expresion$_1$/expression$_2$	expresion. dir = nuevaTemporal expresion. tipo = max(expresion$_1$. tipo, expresion$_2$. tipo) t_1 = ampliar(expresion$_1$. dir, expresion$_1$. tipo, expresion. tipo) t_2 = ampliar(expresion$_2$. dir, expresion$_2$. tipo, expresion. tipo) expresion. code = gen(expresion. dir$' =' t_1'/' t_2$)
expresion → expresion$_1$%expresion$_2$	expresion. dir = nuevaTemporal expresion. tipo = max(expresion$_1$. tipo, expresion$_2$. tipo)

	$t_1 = \text{ampliar}(\text{expresion}_1.\text{dir}, \text{expresion}_1.\text{tipo}, \text{expresion.tipo})$ $t_2 = \text{ampliar}(\text{expresion}_2.\text{dir}, \text{expresion}_2.\text{tipo}, \text{expresion.tipo})$ $\text{expresion.code} = \text{gen}(\text{expresion.dir}' = ' t_1' \% ' t_2)$
expresion → (**expresion**$_1$)	$\text{expresion.dir} = \text{expresion}_1.\text{dir}$ $\text{expresion.tipo} = \text{expresion}_1.\text{tipo}$
expresion → **variable**	Si TS.existe(variable) Entonces expresion.dir = variable.dir expresion.tipo = TS.getTipo(variable) Sino Error("La variable no ha sido declarada") Fin Si
expresion → **num**	expresion.tipo = num.tipo expresion.dir = num.val
expresion → **cadena**	expresion.tipo = cadena.tipo expresion.dir = TablaDeCadenas.add(cadena.val)
expresion → **caracter**	expresion.tipo = caracter.tipo expresion.dir = TablaDeCadens.add(caracter.val)
variable → **id variable_comp**	Si TS.existe(id) Entonces tipo_id = TS.getTipo( ) t = reducir(variable_comp.dir, variable_comp.tipo, tipo_id) Sino Error("El id no ha sido declarado") Fin Si
variable_comp → **dato_est_sim**	variable_comp.dir = dato_est_sim.dir variable.code = dato_est_sim.code
variable_comp → **arreglo**	variable_comp.dir = arreglo.dir variable_comp.base = arreglo.base variable_comp.tipo = arreglo.tipo
variable_comp → (**parametros**)	variable_comp.lista = parametros.lista variable_comp.num = parametros.num
dato_est_sim → **dato_est_sim** . **id**	Si ! TS.existe(id) Entonces STS.getFondo( ).append(id, dir, Tipo) dir ← dir + STT.getFondo( ).getTam(Tipo) Sino Error("El id no ha sido declarado")
dato_est_sim → **ε**	dato_est_sim.dir = null
arreglo → (**expresion**)	t = nuevaTemporal( ) arreglo.dir = nuevaTemporal( ) arreglo.tipo = array arreglo.tam = TT.getTam(expresion.tipo) arreglo.base = expresion.base arreglo.code = gen(t ′ = ′expresion.dir ′ ∗ ′ arreglo.tam)
arreglo → **arreglo**$_1$(**expresion**)	Si TT.getNombre(arreglo_1.tipo) = array Entonces t = nuevaTemporal( ) arreglo.dir = nuevaTemporal( ) arreglo.tipo = TT.getTipoBase(arreglo_1.tipo) arreglo.tam = TT.getTam(arreglo_1.tipo) arreglo.base = arreglo_1.base arreglo.code = gen(t′ = ′expresion.dir′ ∗ ′arreglo.tam) || gen(arreglo.dir ′ = ′ arreglo_1.dir ′ + ′ t)

	Sino Error("La variable asociada no es un arreglo") Fin Si
parametros → lista_param	parametros.lista = lista_param.lista parametros.num = lista_param.num
parametros → ε	parametros.lista = null parametros.num = 0
lista_param → lista_param$_1$, expresion	lista_param.lista = nuevaLista( ) lista_param.lista.append(expresion.tipo) lista_param.num = lista_param$_1$.num + 1
lista_param → expresion	lista_param.lista = nuevaLista( ) lista_param.lista.append(expresion.tipo) lista_param.num = 1