Attribute Grammar

Attributes

Symbol	Attribute Name	Java Type	Inherited/Synthesized	Description
Expression	type	Type	synthesized	Tipo de la expresión
Expression	lvalue	boolean	synthesized	True si se puede colocar a la izquierda de una expresión
Statement	funcion	functionDeclaration	Inherited	Indica la función a la que pertenece la sentencia

Auxiliar Funcitions

Name	Description
sameType(type_a, type_b)	True si son el mismo tipo

Rules

Node	Predicates	Semantic Functions
program ⟶ declaration*		
structDeclaration:declaration ⟶ ID:string variableDeclaration*		
variableDeclaration:declaration ⟶ ID:string type		
functionDeclaration:declaration ⟶ ID:string parameters:variableDeclaration* type? variableDeclaration* statement*	parameters ∈ SimpleType type ∈ SimpleType or type == VOID	Statement(i).funcion = this
print:statement ⟶ expression*	print ∈ SimpleType	
printsp:statement ⟶ expression*	printsp ∈ SimpleType	
println:statement ⟶ expression*	println ∈ SimpleType	
read:statement ⟶ expression	read ∈ SimpleType	

	expression.tipo.lvalue	
if:statement ⟶ expression s1:statement* s2:statement*	Expression.Type == intType	s1.funcion == functionDeclaration s2.funcion == functionDeclaration
while:statement ⟶ expression statement*	Expression.Type == intType	statement == functionDeclaration

return:statement ⟶ expression?	IF return.funcion.Type == VOID
	    expression.Type == null
	ELSE
	    expression.Type == return.funcion.Type	

asignacion:statement ⟶ e1:expression e2:expression	e1 ∈ SimpleType sameType(e1.type , e2.type) e1.lvalue == true	
funcionLlamada:statement ⟶ ID:string expression* [def]functionDeclaration	funcionLlamada.functionDeclaration.parameters.length == expr.length ∀ expression(i).type == funcionLlamada.functionDeclaration.parameters(i).Type	
cast:expression ⟶ targetType:type expression	targetType != expression.type	cast.lvalue = false
structAccess:expression ⟶ expression ID:string	expression.type == structType	structAccess.lvalue = true

arrayAccess:expression ⟶ e1:expression e2:expression	e1.type == arrayType
	e2.type == intType
		arrayAccess.type = e1.type
		arrayAccess.lvalue = true

expresionLlamada:expression ⟶ ID:string expression* [def]functionDeclaration	expression(i).type **==** expresionLlamada.functionDeclaration.parameters(i).Type expresionLlamada.functionDeclaration.Type != null	
not:expression ⟶ expression	not.Type == intType	not.type = expression.type

expresionAritmetica:expression ⟶ e1:expression op:string e2:expression	IF(op == '%')
	    e1.Type == intType
	ELSE
	    e1.Type == intType or e1.Type == realType
	
	sameType(left.value, right.value)
		expresionArithmetica.type = e1.type
		expresionArithmetica.lvalue = false

expresionLogica:expression → e1:expression op:string e2:expression	IF(op == “&&” || op == “||”)
	  e1.Type == intType
	ELSE
	  e1.Type == intType or e1.Type == realType
	
	sameType(e1.value, e2.value)
		expresionLogica.type = intType
		expresionLogica.lvalue = false

variable:expression → ID:string		Variable.type = variable.variableDeclaration.type
		Variable.lvalue = true

litEnt:expression → LITENT:string		litReal.type = realType
		litReal.lvalue = false

litReal:expression → LITREAL:string		litReal.type = realType
		litReal.lvalue = false

litChar:expression → CHAR_LITERAL:string		litReal.type = realType
		litReal.lvalue = false

intType:type → ε		
realType:type → ε		
charType:type → ε		
voidType:type -> ε		
arrayType:type → posicion:string type		
structType:type → nombre:string		

SimpleType = {intType, realType, charType}

Operators samples (cut & paste if needed):
⇒ ⇔ ≠ ∅ ∈ ∉ ∪ ∩ ⊂ ⊄ ∑ ∃ ∀