Attribute Grammar

Nodo	Predicados	Reglas Semánticas
program → *definitions*:definition*		
structField → *name*:String *type*:type		
definitionVariable:definition → *name*:String *type*:type		
definitionStruct:definition → *name*:String *structFields*:structField*		
definitionFunction:definition → *name*:String *definitionFunctionParams*:definitionVariable* *type*:type *localVariables*:definitionVariable* *sentences*:sentence*	type != TypeVoid ⇒ type.primitive definitionFunctionParams.type^{i}.primitive	sentences.fatherFunction^{i}=this
typeInt:type → λ		primitive=true
typeFloat:type → λ		primitive=true
typeChar:type → λ		primitive=true
typeVoid:type → λ		primitive=false
typeArray:type → *size*:expressionConstantInt *type*:type	size.type == TypeInt	primitive=false
typeStruct:type → *name*:String		primitive=false
sentencePrint:sentence → *expression*:expression	(expression ≠ ∅)⇒ expression.type.primitive	
sentencePrintsp:sentence → *expression*:expression	(expression ≠ ∅) ⇒ expression.type.primitive	
sentencePrintln:sentence → *expression*:expression	(expression ≠ ∅) ⇒ expression.type.primitive	
sentenceReturn:sentence → *expression*:expression	(expression == ∅) ⇒ (fatherFunction.type==TypeVoid) (expression != ∅) ⇒ (fatherFunction.type!=TypeVoid && expression.type == fatherFunction.type)	
sentenceRead:sentence → *expression*:expression	expression.type.primitive expression.modifiable	
sentenceAssignment:sentence → *left*:expression *right*:expression	left.type.primitive right.type.primitive left.type == right.type left.modifiable	
sentenceCallFunction:sentence → *name*:String *callFunctionParams*:expression*	|callFunctionParams| == |definition.definitionFunctionParams| callFunctionParams^{i}.type == definition.definitionFunctionParams^{i}.type callFunctionParams.type^{i} .primitive	
sentenceIf:sentence → *condition*:expression *ifSentences*:sentence* *elseSentences*:sentence*	condition.type == TypeInt	ifSentences^{i}.fatherFunction=fatherFunction elseSentences^{i}.fatherFunction=fatherFunction

sentenceWhile:sentence → *condition*:expression *sentences*:sentence*	condition.type == TypeInt	sentences^{i}.fatherFunction= fatherFunction
expressionConstantInt:expression → *value*:String		type=TypeInt modifiable=false
expressionConstantFloat:expression → *value*:String		type=TypeFloat modifiable=false
expressionConstantChar:expression → *value*:String		type=TypeChar modifiable=false
expressionCallFunction:expression → *name*:String *callFunctionParams*:expression*	callFunctionParams.size == definition.definitionFunctionParams.size callFunctionParams^{i}.type == definition.definitionFunctionParams^{i}.type callFunctionParams.type^{i}.primitive definition.type != VoidType	type=definition.type modifiable=false
expressionUnary:expression → *operator*:String *expression*:expression	expression.type == TypeInt	type=TypeInt modifiable=false
expressionCast:expression → *newType*:type *expression*:expression	expression.type.primitive newType.primitive expression.type != newType	type=newType modifiable=false
expressionArithmetic:expression → *left*:expression *operator*:String *right*:expression	left.type == right.type (operator ∈ {“+” , “-“, “*”, “/” }) ⇒ (left.type∈ {TypeInt, TypeFloat} && right.type∈ {TypeInt, TypeFloat}) (operator == “%”) ⇒ (left.type == TypeInt && right.type == TypeInt)	type=left.type modifiable=false
expressionRelational:expression → *left*:expression *operator*:String *right*:expression	left.type == right.type left.type ∈ {TypeInt, TypeFloat} right.type ∈ {TypeInt, TypeFloat}	type=TypeInt modifiable=false
expressionLogical:expression → *left*:expression *operator*:String *right*:expression	left.type == right.type left.type == TypeInt right.type == TypeInt	type=TypeInt modifiable=false
expressionVariable:expression → *name*:String		type=definition.type modifiable=true
expressionStructField:expression → *struct*:expression *name*:String	struct.type.definitionStruct != ∅ struct.type.definitionStruct.field(name) != ∅	type=struct.type.definitionStruct.field(name).type modifiable=true
expressionArray:expression → *array*:expression *index*:expression	index.type == TypeInt array.type.typeOfTheArray != ∅	type=array.type.typeOfTheArray modifiable=true

Atributos

Nodo/Categoría Sintáctica	Nombre del Atributo	Tipo Java	Heredado/Sintetizado	Descripción
type	primitive	boolean	Sintetizado	Indica si el tipo es primitivo
expression	type	Type	Sintetizado	Tipo de la expresión
expression	modifiable	boolean	Sintetizado	Indica si la expresión puede ser modificable (es un lvalue)
sentece	fatherFunction	DefinitionFunction	Heredado	Función en la que se encuentra la sentencia

Métodos Auxiliares

Nodo/Categoría Sintáctica	Método	Tipo de retorno Java	Fase	Descripción
expressionVariable	definition	DefinitionVariable	Identificación	Definición de la variable
sentenceCallFunction	definition	DefinitionFunction	Identificación	Definición de la función
expressionCallFunction	definition	DefinitionFunction	Identificación	Definición de la función
type	definitionStruct	DefinitionStruct		Definición de la estructura si es de tipo estructura, si no, ∅
type	typeOfTheArray	Type		Tipo del array si es de tipo array, si no, ∅
definitionStruct	field(string)	StructField		Campo del struct con el nombre que se pasa como parámetro