## Attribute Grammar

### Attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Attribute Name | Java Type | Inherited/Synthesized | Description |
| expression | **type** | tipo | Synthetized | Type of the expression |
| expression | **lvalue** | boolean | Synthetized | True if the expression can appear to the left of an assignment |
| statement | **padre** | function | Heritated | Se pasa al padre como atributo heredado |
|  |  |  |  |  |

### Rules

|  |  |  |
| --- | --- | --- |
| Node | Predicates | Semantic Functions |
| program → cuerpoPrograma\* | - | - |
| structDefinition:cuerpoPrograma → nombre:string atributos:varDefinition\* metodos:function\* | - | - |
| varDefinition:cuerpoPrograma → declaracion | - | - |
| declaracion → nombre:string tipo:tipo | - | - |
| function:cuerpoPrograma → nombre:string parametros:declaracion\* tipoRetorno:tipo cuerpo:statement\* |  | cuerpo.padre=this(para cada sentencia hay que pasarle como atributo heredado quien es el padre) |
| print:statement → expr\* | Tiene que ser de un tipo simple  Expr.type == intType ||  Expr.type == floatType ||  Expr.type == charType | - |
| printSp:statement → expr\* | Expr.type == intType ||  Expr.type == floatType ||  Expr.type == charType | - |
| printLn:statement → expr\* | Expr.type == intType ||  Expr.type == floatType ||  Expr.type == charType | - |
| asignacion:statement → left:expr right:expr | sameType (left.type, right.type)  left.lvalue == true |  |
| return:statement → expr? | Si(padre.tipoRetorno==voidType){  Expr.isEmpty() ;//Ver que te vacia  }  Expr.tipo == return.padre.tipo |  |
| oneExpr:statement → expr | - | - |
| if:statement → condition:expr thenBlock:statement\* elseBlock:statement\* | Condition.type==intType | thenBlock.padre=this.padre(para cada sentencia que hay en el bloque then)  elseBlock.padre=this.padre (para cada sentencia que hay en el bloque else) |
| while:statement → condition:expr body:statement\* | Condition.type==intType | body.padre=this(para cada sentencia del body) |
| read:statement → expr | Expr.type == intType ||  Expr.type == floatType ||  Expr.type == charType &&  Expr.lvalue(se mira que no se pasen valores que no puedan asignarse) |  |
| stmtVarDefinition:statement → declaracion | - | - |
| stmtFunctionCall**:**statement→ nombre:string expr\* | Expr.length=function.parametros.length  Expr.type= function.parametros.tipo(para cada parámetro y cada expresión) | - |
| fieldAccess:expr → object:expr field:string | object.type==nomType  fieldAcces.structDefinition.atributos.definicion.name == field(Hay que recorrer atributos para comparar el nombre de cada atributo con el que tenemos) | fieldAccess.lvalue = true  fieldAcces.type = fieldAcces.structDefinition.atributos.declaracion.tipo;  MIRAR PORQUE NO DEBE DE TAR BIEN |
| arrayAccess:expr → array:expr index:expr | Index.type==intType  Array.type==arrayType | arrayAccess.type=array.type  arrayAcces.lvalue = true  MIRAR PORQUE NO CREO QUE TE BIEN |
| cast:expr → tipo:tipo target:expr | Tipo!=target.type  Target ¡= array && target ¡=struct  Tipo ¡= arrayType && tipo ¡= nomType | Cast.type=tipo  cast.lvalue = false |
| not:expr → expr | Expr.type==intType | not.type=intType  not.lvalue = false |
| operacionAritmetica:expr → left:expr operador:string right:expr | sameType(left.type, right.type) | operacionAritmetica.type = left.type operacionAritmetica.lvalue = false |
| operacionLogica:expr → left:expr operador:string right:expr | left.type==intType  sameType(left.type, right.type) | operacionLogica.type = left.type operacionLogica.lvalue = false |
| comparacion:expr → left:expr operador:string right:expr | sameType(left.type, right.type) | comparacion.type = intType  arithmetic.lvalue = false |
| functionCall:expr → nombre:string expr\* | Expr.length=function.parametros.length  Expr.type= function.parametros.tipo(para cada parámetro y cada expresión)  functionCall.function.tipoRetorno != void  (no puede devolver void) | Functioncall.type=fucntioncall.function. tipoRetorno  functionCall.lvalue = false |
| parentesis:expr → expr | - | parentesis.type=expr.type  parentesis.lvalue = false |
| variable:expr → nombre:string | - | variable.type = variable.varDefinition.type  variable.lvalue = true |
| intLiteral:expr → intValue:int | - | intLiteral.type = intType intLiteral.lvalue = false |
| realLiteral:expr → floatValue:float | - | realLiteral.type = floatType realLiteral.lvalue = false |
| charLiteral:expr → charValue:char | - | charLiteral.type = charType charLiteral.lvalue = false |
| intType:tipo → ε | - | - |
| floatType:tipo → ε | - | - |
| charType:tipo → ε | - | - |
| voidType:tipo → ε | - | - |
| arrayType:tipo → tipoBase:tipo intValue:int | - | - |
| nomType:tipo → nombre:string | - | - |

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