**Attribute Grammar**

**Attributes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Attribute Name | Java Type | Inherited/Synthesized | Description |
| expression | lvalue | Boolean | Synthesized | True if it can be the left side of the assignment |
| expression | Type | Type | Synthesized | Type of the expresión |
| Statement | functionDefinition | functionDefinition | Inherited | The definition of the function to call |
| structAccess | structField | structField | Inherited | The definition of the struct |

**Auxiliary Functions**

|  |  |
| --- | --- |
| Name | Description |
| sameType(type\_a, type\_b) | True if both params are equal |
| getStructFieldType(field:string) | Returns the type of the field from the struct |
|  |  |

**Rules**

|  |  |  |
| --- | --- | --- |
| Node | Predicates | Semantic Functions |
| **program** → definition\* |  |  |
| **varDefinition**:definition → name:string type |  |  |
| **structDefinition**:definition → name:string structField\* |  |  |
| **functionDefinition**:definition → name:string varDefinition\* type definition\* statement\* | ∀varDefinition ⇒ isSimpleType(varDefinition.type)  isSimpleType(type) || type == VoidType | ∀statement ⇒ statement.functionDefinition = this |
| **structField** → name:string type | isSimpleType(type) |  |
| **assignment**:statement → left:expression right:expression | sameType(left.type, right.type)  simpleType(left.type) |  |
| **functionCallStatement**:statement → name:string expression\* | sameParams(expression\*, functionCallStatement.functionDefinition.params) |  |
| **if**:statement → condition:expression ifBody:statement\* elseBody:statement\* | Condition.type == IntType | ∀ifBody.functionDefinition = this  ∀elseBody.functionDefinition = this |
| **while**:statement → condition:expression statement\* | Condition.type == IntType | ∀ statement.functionDefinition = this |
| **read**:statement → expression | isSimpleType(expression) |  |
| **print**:statement → expression\* lexema:string | ∀expression\* ⇒ isSimpleType(expression) |  |
| **return**:statement → expression? | If (expression != null) sameType(return.functionDefinition.type, expression.type)  else sameType(return.functionDefinition.type, VoidType) |  |
| **intLiteral**:expression → value:int |  | Type = new IntType()  Lvalue = false |
| **floatLiteral**:expression → value:float |  | Type = new FloatType()  Lvalue = false |
| **charLiteral**:expression → value:string |  | Type = new CharType()  Lvalue = false |
| **variable**:expression → name:string |  | Type = variable.definition.type  Lvalue = true |
| **arithmetic**:expression → left:expression operator:string right:expression | sameType(left.type, right.type) | Type = left.type  Lvalue = false |
| **arithmeticComparison**:expression → left:expression operator:string right:expression | If(operator == ‘%’) left.type == IntType && right.type == IntType  Else intOrDouble(left.type, right.type) | Type = left.type  Lvalue = false |
| **logicalComparison**:expression → left:expression operator:string right:expression | right.type == IntType && left.type == IntType | Type = left.type  Lvalue = false |
| **negation**:expression → expression | right.type == IntType && left.type == IntType | Type = expression.type  Lvalue = false |
| **functionCallExpression**:expression → name:string expression\* | sameParams(expression\*, functionCallExpression.definition.params) | Type =functionCallExpression.definition.type  Lvalue = false |
| **structAccess**:expression → expression field:string | structAccess.definition.fields ⊂ field | Type = getStructFieldType(field)  Lvalue = true |
| **cast**:expression → type expression | isSimpleType(type) | Type = type  Lvalue = false |
| **arrayAccess**:expression → left:expression right:expression | right.type == IntType | Type = left.definition.type  Lvalue = true |
| **intType**:type → ε |  |  |
| **floatType**:type → ε |  |  |
| **charType**:type → ε |  |  |
| **varType**:type → name:string |  |  |
| **arrayType**:type → size:int type |  |  |
| **structType**:type → name:string |  |  |
| **voidType**:type → ε |  |  |
| **errorType**:type → ε |  |  |

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