## Attribute Grammar – UO288787

### Attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Attribute Name | Java Type | Inherited/Synthesized | Description |
| Expression | type | Type | Synthetic | The type of the expression |
| Expression | lvalue | Boolean | Synthetic | Wether the expression can be the assigned in the assignment or not |
| Statement | definition | functionDefinition | Inherited | Lets us know what function the statement is defined in |

### Auxiliary Functions

|  |  |
| --- | --- |
| Name | Description |
| sameType(typeA,typeB) | Returns true if types are equal |
| isPrimitive(type) | Returns true if type is integer, float or character |
| isNum(type) | Returns true if type is integer or float |

### Rules

|  |  |  |
| --- | --- | --- |
| Node | Predicates | Semantic Functions |
| program → definition\* |  |  |
| varDefinition:definition → name:string type |  | varDefinition.type = type  varDefinition.lvalue = false |
| structDefinition:definition → name:string field\* |  | structDefinition.type = StructType  structDefinition.lvalue = false |
| functionDefinition:definition → name:string varDefinition\* type? definition\* statement\* | If (type!=null)  ifPrimitive(type)  isPrimitive(varDefinition.type) | functionDefinition.type = type  functionDefinition.lvalue = false  statement.function = functionDefinition |
| field → name:string type |  | field.lvalue = true  field.type = type |
| print:statement → expression | isPrimitive(expression.type) | (All statements have lvalue false and no type (Error)) |
| read:statement → expression | isPrimitive(expression.type) |  |
| functionCallStatement:statement → name:string expression\* |  |  |
| assignment:statement → left:expression right:expression | sameType(left.type, right.type)  isPrimitive(left.type)  left.lvalue == true |  |
| conditional:statement → expression ifStatements:statement\* elseStatements:statement\* | Expression.type == IntType | ifStatements.function =conditional.function  elseStatements.function = conditional.function |
| while:statement → expression loopStatements:statement\* | Expression.type = IntType | Statement.function = conditional.function |
| return:statement → expression? | Return.type == return.function.type |  |
| intType:type → ε |  | (Types do nothing) |
| floatType:type → ε |  |  |
| charType:type → ε |  |  |
| arrayType:type → intValue:int type |  |  |
| structType:type → name:string |  |  |
| variable:expression → name:string |  | variable.lvalue = true  variable.type = variable.definition.type |
| intLiteral:expression → intValue:int |  | intLiteral.lvalue = false  intLiteral.type = IntLiteral |
| floatLiteral:expression → floatValue:float |  | floatLiteral.lvalue = false  floatLiteral.type = FloatType |
| charLiteral:expression → charValue:char |  | charLiteral.lvalue = false  charLiteral.type = CharType |
| functionCallExpression:expression → name:string expression\* |  | functionCallExpression.lvalue = false  functionCallExpression.type = functionCallExpression.definition.type |
| structAccess:expression → expression name:string |  | structAccess.lvalue = true  structAccess.type = |
| arrayAccess:expression → left:expression right:expression | Right.getType() instanceof IntType | arrayAccess.lvalue = true  arrayAccess.type = left.type |
| cast:expression → type expression | !sameType(expression.getType(), type) | cast.type = type  arithmetic.lvalue=false |
| arithmetic:expression → left:expression operator:string right:expression | operator.equals(‘+-\*/’) && left.type == IntType && right.type == IntType  operator.equals(‘%’)  && left.type == FloatType && right.type == FloatType | arithmetic.lvalue = false  arithmetic.type = left.type |
| comparison:expression → left:expression operator:string right:expression | isNum(left.type) && isNum(right.type) | comparison.type = IntType  comparison.lvalue = false |
| logicBinary:expression → left:expression operator:string right:expression | isNum(left.type) && isNum(right.type) | logicBinary.type = IntType  logicBinary.lvalue = false |
| logicUnary:expression → left:expression operator:string right:expression | isNum(left.type) | logicUnary.type = IntType  logicUnary.lvalue = false |
|  |  |  |

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