Annot ::= “@” Id [“(” { AnnotParam } “)”]
AnnotParam ::= IntValue | StringValue | Id
AssertStat ::= “assert” Expression “,” StringValue
AssignExpr ::= Expression [“=” Expression]
BasicType ::= “Int” | “Boolean” | “String”
BasicValue ::= IntValue | BooleanValue | StringValue
BooleanValue ::= “true” | “false”
ClassDec ::= [“open”] “class” Id [“extends” Id] MemberList “end”
CompStatement ::= “{” { Statement } “}”
Digit ::= “0” | ... | “9”
Expression ::= SimpleExpression [Relation SimpleExpression]
ExpressionList ::= Expression { “,” Expression }
Factor ::= BasicValue | “(” Expression “)” | “!” Factor | “nil” | ObjectCreation | PrimaryExpr
FieldDec ::= “var” Type IdList [“;”]
FormalParamDec ::= ParamDec { “,” ParamDec }
HighOperator ::= “∗” | “/” | “&&”
IdList ::= Id { “,” Id }
IfStat ::= if” Expression “{” Statement “}” [“else” “{” Statement “}”]
IntValue ::= Digit { Digit }
LocalDec ::= “var” Type IdList [“=” Expression]
LowOperator ::= +” | “−” | “||”
MemberList ::= { [Qualifier] Member }
Member ::= FieldDec | MethodDec
MethodDec ::= “func” IdColon FormalParamDec [“->” Type] “{” StatementList “}” |
“func” Id [“->” Type] “{” StatementList “}”
ObjectCreation ::= Id “.” “new”
ParamDec ::= Type Id
Program ::= { Annot } ClassDec { { Annot } ClassDec }
Qualifier ::= “private” “public” “override” “override” “public” “final” “final” “public” “final”
“override” “final” “override” “public” “shared” “private” “shared” “public”
ReadExpr ::= “In” “.” (“readInt” | “readString”)
RepeatStat ::= “repeat” StatementList “until” Expression
PrimaryExpr ::= “super” “.” IdColon ExpressionList | “super” “.” Id |Id | Id “.” Id | Id “.”
IdColon ExpressionList | “self” | “self” “.” Id | “self” “.” IdColon ExpressionList | “self” “.” Id
“.” IdColon ExpressionList | “self” “.” Id “.” Id | ReadExpr
Relation ::= “==” | “<” | “>” | “<=” | “>=” | “! =”
ReturnStat ::= “return” Expression
Signal ::= “+” | “−”
SignalFactor ::= [Signal] Factor
SimpleExpression ::= SumSubExpression { “++” SumSubExpression }

SumSubExpression ::= Term { LowOperator Term }
Statement ::= AssignExpr “;” | IfStat | WhileStat | ReturnStat “;” | WriteStat “;” | “break” “;” | “;” | RepeatStat “;” | LocalDec “;” | AssertStat “;”
StatementList ::= { Statement }
Term ::= SignalFactor { HighOperator SignalFactor }
Type ::= BasicType | Id
WriteStat ::= “Out” “.” [“print:” | “println:”] Expression
WhileStat ::= “while” Expression “{” StatementList “}”\