Syntax :

program = compundStatement

newlineSeparator = “\n”

factor = identifier | constant | “(” arithmetichExpression “)”

term = factor [ ( “\*” |”/” | “%” ) term ]

arithmeticExpression = term [ ( “+” | “-” ) arithmeticExpression ]

negation = “!”

booleanOperator = “&&” | “||”

booleanTerm = [ negation ] ( identifier | booleanConstant | “(” booleanExpression “)” )

relationalOperator = “<” | “<=” | “=” | “>” | “>=”

booleanExpression = booleanTerm [ booleanOperator booleanExpression ] |

arithmeticExpression relationalOperator arithmeticExpression

expression = arithmeticExpression | booleanExpression

readArgument = identifier

writeArgument = expression

ioStatement = “read” “(” readArgument “)” | “write” “(” ( writeArgument ) “)”

primitiveType = “Integer” | “Character” | “String” | “Boolean”

arrayType = primitiveType”[” arithmeticExpression “]”

type = primitiveType | arrayType

declarationStatement = “val” identifier “:” type

assignment = “=” expression

assignmentStatement = identifier assignment

initializationShorthandStatement = declarationStatement assignment

ifStatement = “if” “(” booleanExpression “)” statement [ “else” statement ]

whileStatement = “while” “(” booleanExpression “)” statement

forStatement = “for” “(” constant “to” constant “)” statement

controlFlowStatement = ifStatement | whileStatement | forStatement

simpleStatement = declarationStatement | assignmentStatement | initializationShorthandStatement | . constrolFlowStatement

simpleStatementSequence = simpleStatement { newlineSeparator simpleStatement } /\* doesn’t end with newline \*/

compundStatement = “{” mixedStatement { newlineSeparator mixedStatement } “}” /\* doesn’t end with newline /\*

mixedStatement = simpleStatemenentSequence | compunedStatement /\* doesn’t end with newline \*/

statement = simpleStatement | compundStatement