PROGRAM	STATEMENT ; (STATEMENT 😉*
STATEMENT	'var' IDENTIFIER : TYPE (:= EXPRESSION) ;
	IDENTIFIER := EXPRESSION ;
	for IDENTIFIER in EXPRESSION .. EXPRESSION do STATEMENT ;+ end for
	read IDENTIFIER
	print EXPRESSION
	assert '(' EXPRESSION ')'
EXPRESSION	OPERAND OPERATOR OPERAND
	UNARY_OPERAND? OPERAND
OPERAND	INTEGER
	STRING
	IDENTIFIER
	'(' EXPRESSION ')'
TYPE	INTEGER
	STRING
	BOOLEAN
IDENTIFIER	'some identifier user has defined'
RESERVED_KEYWORD	VAR
	FOR
	END
	IN
	DO
	READ
	PRINT
	INT
	STRING
	BOOL
	ASSERT
OPERATOR	+

	-
	*
	/
	<
	=
	&
	!

Limited grammar for calculations:

EXPRESSION	BINARY
	UNARY
	LITERAL
	GROUPING
LITERAL	NUMBER
	STRING
	TRUE
	FALSE
GROUPING	'(' EXPRESSION ')' ;
UNARY	- EXPRESSION
	! EXPRESSION
BINARY	EXPRESSION OPERATOR EXPRESSION
OPERATOR	+
	-
	*
	/
	<
	=
	&
	!

Unambiguous grammar:

expression	equality ;
equality	comparison (("!=" | "=") comparison)* ;
comparison	term ((">" | ">=" | "<" | "<=") term)* ;
term	factor (("-" | "+") factor)* ;
factor	unary (("/" | "*”) unary)* ;
unary	("!" | "-") unary
	primary ;
primary	NUMBER | STRING | "true" | "false" | "null"
	"(" expression ")" | IDENTIFIER ;

Grammar to enable statements:

program	statement* EOF ;
statement	expressionStatement ;
	printStatement ;
expressionStatement	expression ";" ;
printStatement	"print" expression ";" ;

Grammar to enable name declarations:

program	declaration* EOF ;
declaration	variableDeclaration ;
	statement ;
statement	expressionStatement ;
	printStatement ;
variableDeclaration	"var" IDENTIFIER (":=" expression)? ";" ;

Assignment:

expression	assignment
assignment	IDENTIFIER ":=" assignment
	equality