FINAL CONTEXT FREE GRAMMAR
B++ Programming Language

The grammar **G = (V, T, S, P)** of our interpreter is defined by:

V = { <source>, <block>, <statement>, <expression>, <assignment>, <ifStatement>, <forStatement>, <whileStatement>, <doWhileStatement>, <missingSemiColonAss>, <ifStat> <elseIfStat>, <elseStat>, <missingLBraceForLoop> <missingRBraceForLoop>, <missingLBraceWhileLoop>, <missingRBraceWhileLoop>, <missingRBraceDoWhileLoop>, <missingLBraceDoWhileLoop>, <missingSemiColonloop>, <functionCall>, <idList>, <exprList>, <expression>, <indexes>, <list>, <PRINTLN>, <INPUT>, <IF>, <ELSE>, <FOR>, <WHILE>, <TO>, <DO>, <NULL>, <IDENTIFIER>, <NUMBER>, <STRING>, <letters+>, <digits+>}
T = {a, b, .., z, A, B, .., Z, 0, 1, .., 9, ;, :, {, }, [,], ,, ‘, “, +, -, *, /, %, &, |, !} ∪{*other special characters*}
S = <source>
P has the following productions:

Production for the program

<source> → <block> EOF

<block> → (<statement>)* (RETURN <expression> SCOLON)?

Productions for variable declaration

<assignment> → VAR IDENTIFIER <indexes>? ‘=’ <expression>
→ IDENTIFIER <indexes>? ‘=’ <expression>

Productions for statements

<statement> → <assignment> SCOLON
→ <functionCall> SCOLON
→ <ifStatement>
→ <forStatement>
→ <whileStatement>
→ <doWhileStatement>
→ <missingSemiColonAss>

Productions for expressions

<expression> → ‘-‘ <expression>
→ ‘!‘ <expression>
→ <assoc=right> <expression> ‘^’ <expression>
→ <expression> <op>=(‘*’ | ‘/’ | ‘%’) <expression>
→ <expression> <op>=(‘+’ | ‘-’) <expression>
→ <expression> (‘++’ | ‘**’ | ‘//’ | ‘%%’ | ‘--') <expression>
→ <expression> <op>=(‘>=’ | ‘<=’ | ‘>’ | ‘<’) <expression>

→ <expression> <op>=('==' | '!=') <expression>
→ <expression> '&&' <expression>
→ <expression> '||' <expression>
→ NUMBER
→ NULL
→ BOOL
→ STRING <indexes>?
→ IDENTIFIER <indexes>?
→ '(' <expression> ')' <indexes>?
→ <expression> ')'
→ '(' <expression>
→ INPUT '(' STRING? ')'

Other Productions

<ifStatement> → <ifStat> <elseIfStat>* <elseStat>?
<ifStat> → IF <expression> LBRACE <block> RBRACE
<elseIfStat> → ELSE IF <expression> LBRACE <block> RBRACE
<elseStat> → ELSE LBRACE <block> RBRACE
<forStatement> → FOR IDENTIFIER '=' <expression> TO expression LBRACE block RBRACE
→ <missingLBraceForLoop>
→ <missingRBraceForLoop>
<missingLBraceForLoop> → FOR IDENTIFIER '=' <expression> TO expression block RBRACE
<missingRBraceForLoop> → FOR IDENTIFIER '=' <expression> TO expression LBRACE block
<WhileStatement> → WHILE <expression> LBRACE block RBRACE
→ <missingLBraceWhileLoop>
→ <missingRBraceWhileLoop>
<missingLBraceWhileLoop> → WHILE <expression> block RBRACE
<missingRBraceWhileLoop> → WHILE <expression> LBRACE block
<missingSemiColonAss> → <assignment>
→ <functionCall>
<DoWhileStatement> → DO LBRACE <block> RBRACE WHILE <expression> SCOLON
→ <missingLBraceDoWhileLoop>
→ <missingRBraceDoWhileLoop>
→ <missingSemiColonLoop>
<missingLBraceDoWhileLoop> → DO <block> RBRACE WHILE <expression> SCOLON
<missingRBraceDoWhileLoop> → DO LBRACE <block> WHILE <expression> SCOLON
<missingSemiColonLoop> → DO LBRACE <block> RBRACE WHILE <expression>
<functionCall> → IDENTIFIER '(' exprList? ')'
→ PRINTLN '(' expression? ')'
→ PRINT '(' expression? ')'
→ PRINTLN expression? ')'
→ PRINTLN '(' expression?
<exprList> → <expression> (',' expression)*
<indexes> → ('[' <expression> ']')+

Keywords

<PRINTLN>	→	‘brint’
<INPUT>	→	‘bnput’
<IF>	→	‘bf’
<ELSE>	→	‘blse’
<FOR>	→	‘bor’
<RETURN>	→	‘beturn’
<WHILE>	→	‘bhile’
<TO>	→	‘bto’
<DO>	→	‘bdo’
<NULL>	→	‘bull’
<BOOL>	→	‘brue’
	→	‘balse’
<NUMBER>	→	INT (‘.’ Digit *)?
<IDENTIFIER>	→	[a-zA-Z_] [a-zA-Z_0-9]*