Falak BNP → LL(1)

(0) <program> ::= <def-list> "EOF"

(1) <def-list> ::= (<def>)*

(2) <def> ::= <var-def> | <fun-def>

(3) <fun-def> ::= <id> (<param-list>) { <var-def-list> <stmt-list> }

(4) <var-def-list> ::= (<var-def>)*

(5) <var-def> ::= var <var-list> ;

(6) <var-list> ::= <id-list>

(7) <param-list> ::= (<id-list>)?

(8) <id-list> ::= <id> <id-list-cont>

(9) <id-list-cont> ::= (, <id>)*

(10) <stmt-list> ::= (<stmt>)*

(11) <stmt> ::= <stmt-assign> | <stmt-incr> | <stmt-decr> | <stmt-fun-call> | <stmt-if> | <stmt-while> | <stmt-do-while> | <stmt-break> | <stmt-return> | <stmt-empty>

(12) <stmt-assign> ::= <id> = <expr> ;

(13) <stmt-incr> ::= inc <id> ;

(14) <stmt-decr> ::= dec <id> ;

(15) <stmt-fun-call> ::= <fun-call> ;

(16) <fun-call> ::= <id> (<expr-list>)

(17) <expr-list> ::= (<expr> <expr-list-cont>)?

(18) <expr-list-cont> ::= (, <expr>)*

(19) <stmt-if> ::= if (<expr>) { <stmt-list> } <elsе-if-list> <elsе>

(20) <elsе-if-list> ::= (elseif (<expr>) { <stmt-list> })*

(21) <elsе> ::= (elsе { <stmt-list> })?

(22) <stmt-while> ::= while (<expr>) { <stmt-list> }

(23) <stmt-do-while> ::= do { <stmt-list> } while (<expr>) ;

(24) <stmt-break> ::= break ;

(25) <stmt-return> ::= return <expr> ;

(26) <stmt-empty> ::= ;

(27) <expr> ::= <expr-or>

(28) <expr-or> ::= <expr-and> (<op-or> <expr-and>)*

(29) <op-or> ::= || | ^

(30) <expr-and> ::= <expr-comp> (&& <expr-comp>)*

(31) <expr-comp> ::= <expr-rel> (<op-com> <expr-rel>)*

(32) <op-comp> ::= == | !=

(33) <expr-rel> ::= <expr-add> (<op-rel> <expr-add>)*

(34) <op-rel> ::= < | <= | > | >=

(35) <expr-add> ::= <expr-mul> (<op-add> <expr-mul>)*

(36) <op-add> ::= + | -

(37) <expr-mul> ::= <expr-unary> (<op-mul> <expr-unary>)*

(38) <op-mul> ::= * | / | %

(39) <expr-unary> ::= (<op-unary>)* <expr-primary>

(40) <op-unary> ::= + | - | !

(41) <expr-primary> ::= <id> | <fun-call> | <array> | <lit> | (<expr>)

(42) <array> ::= [<expr-list>]

(43) <lit> ::= <lit-bool> | <lit-int> | <lit-char> | <lit-str>