Grammatik (läsbar)

START → FUNCDEF START'
START' → START | ε

FUNCDEF → TYPE id (PARS) BLOCK

TYPE → int | void

PARS → int id PARS' | void | ε
PARS' → , int id PARS' | ε

BLOCK → { BLOCK' }
BLOCK' → STMT BLOCK' | ε

STMT → IDENT ; | IFSTMT ; | WHILESTMT ; | read (id) ; | write EXPR ; | VARDEC ; | return
EXPR ;

IDENT → id IDENT'
IDENT' → ASSIGN | FUNCCALL
ASSIGN → = EXPR
FUNCCALL → (ARGS)

IFSTMT → if (EXPR) BLOCK IFSTMT'
IFSTMT' → else IFSTMT" | ε
IFSTMT" → BLOCK | IFSTMT

WHILESTMT → while (EXPR) BLOCK

VARDEC → int id VARDEC'
VARDEC' → = EXPR | , id VARDEC' | ε

ARGS → EXPR ARGS' | ε
ARGS' → , EXPR ARGS' | ε

EXPR → SUM EXPR'
EXPR' → == EXPR | < EXPR | <= EXPR | > EXPR | >= EXPR | ε

SUM → TERM SUM'
SUM' → + SUM | - SUM | ε

TERM → num TERM' | id TERM" | (EXPR) | !TERM
TERM' → * TERM | / TERM | ε
TERM" → FUNCCALL | TERM'

Grammatik (implementerbar)

START_1 → TYPE id (PARS_1) BLOCK_1 START_2
START_2 → START_1 | ε

TYPE → int | void

PARS_1 → int id PARS_2 | void
PARS_2 → , int id PARS_2 | ε

BLOCK_1 → { BLOCK_2 | STMT
BLOCK_2 → STMT BLOCK_2 | }

STMT → id IDENT ; | if (EXPR_1) BLOCK_1 IFSTMT_1 | while (EXPR_1) BLOCK_1
          | read id ; | write EXPR_1 ; | int id VARDEC ; | return EXPR_1 ;

IDENT → = EXPR_1 | (ARGS_1)

IFSTMT_1 → else IFSTMT_2 | ε
IFSTMT_2 → if (EXPR_1) BLOCK_1 IFSTMT_1 | BLOCK_1

VARDEC → , id VARDEC | ε

ARGS_1 → EXPR_1 ARGS_2
ARGS_2 → , EXPR_1 ARGS_2 | ε

EXPR_1 → EXPR_2 EXPR_3
EXPR_2 → SUM_1 EXPR_4
EXPR_3 → == EXPR_1 | ε
EXPR_4 → < EXPR_2 | <= EXPR_2 | ε

SUM_1 → TERM_1 SUM_2
SUM_2 → + SUM_1 | - SUM_1 | ε

TERM_1 → num TERM_2 | id TERM_3 | (EXPR _1) | ! NOTOPERAND_1 TERM_2
TERM_2 → * TERM_1 | / TERM_1 | ε
TERM_3 → (ARGS_1) TERM_2 | TERM_2

NOTOPERAND_1 → (EXPR_1) | ! NOTOPERAND_1 | num | id NOTOPERAND_2
NOTOPERAND_2 → (ARGS_1) | ε