LL(l) Grammar

<program> → <decl_list>

<decl_list> → <decl> ';' <decl_list> | λ

<decl> → var id ':' <type>
| constant id ' =' <expr>
| function id <formal_list> ':' <type> <block>
| procedure id <formal_list> <block>

<type> → integer | boolean | string

<formal_list> → '(' < formal_list_tail>

< formal_list_tail> → ')' | < formals> ')'

<formals> → id ':' <type> < formals_tail>

< formals_tail> → ',' id ':' <type> < formals_tail> | λ

<stmt> → id <id_tail>
| if <expr> then <stmt> <stmt_tail>
| while <expr> do <stmt> od
| for id ':=' <expr> to <expr> do <stmt> od
| read'('id')'
| write' ('id ')'
| return '(' <expr>')'
| <block>

<id_tail> →' :=' <expr>|<arg-list>

<stmt_tail > → fi | else <stmt> fi

< block> → begin <var _decl_list> <stmt _list> end

<var _decl_list> → <var _decl> ';' < var _decl_list> | λ

<var _decl> → var id ':' <type>

<stmt_list> → <stmt> ';' <stmt_list> | λ

<arg_list> → '(' <arg_list_tail>

<arg_list_tail> → ')' | <args> ')'

<args> → <expr> <args_tail>

<args_tail> → ',' <args> | λ

<expr> → <expr1> <expr_tail>

<expr_tail> → *<rel_conj> <expr1> <expr_tail>* | *λ*

<expr1> → *<expr2> <expr1_tail>*

<expr1_tail>→ *<rel_op> <expr2> <expr1_tail>* | *λ*

<expr2> → *<expr3> <expr2_tail>*

<expr2_tail> → *<arith_op_tail1> <expr3> <expr2_tail>* | *λ*

<expr3> → *<expr4><expr3_tail>*

<expr3_tail> → *<arith_op_tail2> <expr4> <expr3_tail>* | *λ*

<expr4> → *<unary_op> <expr4>* | *<expr5>*

<expr5> → *id <expr_id_tail>* | *intger* |*string* | *true* |*false* |
'('<expr>') '

<expr*_id_tail*> → *<arg_list>* | λ

<arith_op_tail1> → *'+'* | *'-'*

<arith_op_tail2> →'*' | '/'

<rel_op> → '='|'!=' |'<' |'<=' |'>' |'>='

<rel_conj> → ***and*** | ***or***

<unary_op>→ **-** | ***not***