LL grammar

<prog> -> <decl-list> Scope EOL <stat-list> End Scope EOL EOF
<decl-list> -> <var-decl> <decl-list>
<decl-list> -> <func-decl> <decl-list>
<decl-list> -> <func-def> <decl-list>
<decl-list> -> <empty_statement>
<decl-list> -> ε
<func-decl> -> Declare Function id (<param-list>) As <type> EOL
<func-def> -> Function id (<param-list>) As <type> EOL <stat-list> End Function EOL
<var-decl> -> Dim <scope-modifier-list> id As <type> <opt-initialiser> EOL
<var-decl> -> Static id as <type> <opt-initialiser> EOL
<param-list> -> <param> <param-list-cont>
<param-list> -> ε
<param> -> id As <type>
<param-list-cont> -> , <param> <param-list-cont>
<param-list-cont> -> ε
<scope-modifier-list> -> <scope-modifier> <scope-modifier-list>
<scope-modifier-list> -> ε
<scope-modifier> -> Shared
<opt-initialiser> -> = <expr>
<opt-initialiser> -> ε
<type> -> Integer
<type> -> Double
<type> -> String
<stat-list> -> <statement> <stat-list>
<stat-list> -> ε
<statement> -> <assignment>
<statement> -> <read-statement>
<statement> -> <print-statement>
<statement> -> <selection-statement>
<statement> -> <iteration-statement>
<statement> -> <return-statement>
<statement> -> <var-decl>
<statement> -> <iteration-control-statement>
<statement> -> <empty_statement>
<assignment> -> id = <expr> EOL
<read-statement> -> Input id EOL
<print-statement> -> print <expr> ; <expr-list> EOL
<expr-list> -> <expr> ; <expr-list>
<expr-list> -> ε
<selection-statement> -> If <expr> then EOL <stat-list> <alternative-statement> End If EOL
<alternative-statement> -> elseif <expr> then EOL <stat-list> <alternative-statement>
<alternative-statement> -> else EOL <stat-list>
<alternative-statement> -> ε
<iteration-statement> -> for id <opt-type-decl> = <expr> to <expr> <opt-step> EOL <stat-list> Next
<opt-id> EOL
<opt-type-decl> -> as <type>

<opt-type-decl> -> ε
<opt-step> -> step <expr>
<opt-step> -> ε
<opt-id> -> id
<opt-id> -> ε
<iteration-statement> -> do <do-cycle> EOL
<do-cycle> -> EOL <stat-list> Loop <opt-cond>
<do-cycle> -> <cond> EOL <stat-list> Loop
<opt-cond> -> <cond>
<opt-cond> -> ε
<cond> -> <do-mode> <expr>
<do-mode> -> until
<do-mode> -> while
<return-statement> -> return <expr> EOL
<iteration-control-statement> -> <control-statement> <cycle-type> <cycle-type-list> EOL
<control-statement> -> Exit
<control-statement> -> Continue
<cycle-type-list> -> , <cycle-type> <cycle-type-list>
<cycle-type-list> -> ε
<cycle-type> -> for
<cycle-type> -> do
<empty_statement> -> EOL