LL grammar

<prog> -> <decl\_list> Scope EOL <stat\_list> End Scope 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 <opt\_scope\_modifier> 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  
<opt\_scope\_modifier> -> Shared  
<opt\_scope\_modifier> -> ε  
<opt\_initialiser> -> = <expr>  
<opt\_initialiser> -> ε  
<type> -> Integer  
<type> -> Double  
<type> -> String<cycle\_stat\_list> -> <statement> <cycle\_stat\_list>  
<cycle\_stat\_list> -> <iteration\_control\_statement> <cycle\_stat\_list>  
<cycle\_stat\_list> -> ε  
<stat\_list> -> <statement> <stat\_list><stat\_list> -> ε<statement> -> <assignment>  
<statement> -> <read\_statement>  
<statement> -> <print\_statement>  
<statement> -> <scope\_statement>  
<statement> -> <selection\_statement>  
<statement> -> <iteration\_statement>  
<statement> -> <return\_statement>  
<statement> -> <empty\_statement>  
<statement> -> <var\_decl>  
<assignment> -> id = <expr> EOL  
<read\_statement> -> Input id EOL  
<print\_statement> -> print <expr> ; <expr\_list> EOL  
<expr\_list> -> <expr> ; <expr\_list>  
<expr\_list> -> ε  
<scope\_statement> -> Scope EOL <stat\_list> End Scope EOL  
<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 <cycle\_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 <cycle\_stat\_list> Loop <opt\_cond>  
<do\_cycle> -> <cond> EOL <cycle\_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