Chimera Syntax Analysis

EBNF – Notation

Program	→	Declaration program statement* end ;
Declaration	→	[const constant-declaration +] [var variable-declaration +] procedure-declaration*
constant-declaration	→	identifier := literal ;
variable-declaration	→	identifier (, identifier) * : type ;
literal	→	simple-literal | list
simple-literal	→	integer-literal | string-literal | boolean-literal
type	→	simple-type | list-type
simple-type	→	integer | string | boolean
list-type	→	list of simple-type
list	→	{ [simple-literal (, simple-literal)*] }
procedure-declaration	→	procedure identifier (parameter-declaration *) [: type] ; [const constant-declaration +] [var variable-declaration +] begin statement* end ;
parameter-declaration	→	identifier (, identifier) * : type ;
statement	→	assignment-statement | call-statement | if-statement | loop-statement | for-statement | return-statement | exit-statement
assignment-statement	→	identifier [[expression]] := expression ;
call-statement	→	identifier ([expression (, expression) *]) ;
if-statement	→	if expression then statement * (else if expression then statement *) * [else statement *] end ;
loop-statement	→	loop statement end ;
for-statement	→	for identifier in expression do statement * end ;
return-statement	→	return [expression] ;
exit-statement	→	exit ;
expression	→	logic-expression
logic-expression	→	relational-expression (logic-operator relational-expression) *
logic-operator	→	and | or || xor
relational-expression	→	sum-expression (relational-operator sum-expression) *
relational-operator	→	= | <> | < | > | <= | >=
sum-expression	→	mul-expression (sum-operator mul-expression) *

```
sum-operator        →   + | -
mul-expression      →   unary-expression ( mul-operator unary-expression ) *
mul-operator        →   * | div | rem
unary-expression    →   not unary-expression
                        | - unary-expression
                        | simple-expression
simple-expression   →   ( ( expression ) | call | identifier | literal )
                        [ [ expression ] ]
call                →   identifier ( [ expression ( , expression ) * ] )
```
