Grammar

```
program → declaration-list
declaration-list → declaration {declaration}
declaration → int ID declaration' | void ID fun-declaration'
declaration' → var-declaration' | fun-declaration'
var-declaration → int ID [ [ NUM ] ] ;
var-declaration' → [ [ NUM ] ] ;
fun-declaration' → ( params ) compound-stmt
params → param-list | void
param-list → param { , param }
param → int ID [ [ ] ]
compound-stmt → { local-declarations statement-list }
local-declarations → {var-declaration}
statement-list → {statement}
statement → expression-stmt | compound-stmt | selection-stmt
            | iteration-stmt | return-stmt
expression-stmt → [expression] ;
selection-stmt → if ( expression ) statement [ else statement ]
iteration-stmt → while ( expression ) statement
return-stmt → return [expression] ;

expression → ( expression ) simple-expression' | NUM simple-expression'
            | ID expression'
expression' → = expression | ( args ) simple-expression'
            | [ expression ] expression'' | simple-expression'
expression'' → = expression | simple-expression'
var → [ [ expression ] ]
simple-expression' → additive-expression' [ relop additive-expression ]
relop → <= | < | > | >= | == | !=
additive-expression → term { addop term }
additive-expression' → term' { addop term }
addop → + | -
term → factor { mulop factor }
term' → { mulop factor }
mulop → * | /
factor → ( expression ) | ID varcall | NUM

varcall → var | call
call → ( args )
args → [ arg-list ]
arg-list → expression { , expression }
```