Grammar:

```
// List of Colons

semi_colon_list
   : ';' semi_colon_list_tail

semi_colon_list_tail
   : semi_colon_list
   | ε

---

// Assignment Operators

assignment_operator //Code Gen = first. Add more later
   : '='
   | MUL_ASSIGN
   | DIV_ASSIGN
   | MOD_ASSIGN
   | ADD_ASSIGN
   | SUB_ASSIGN
   | LEFT_ASSIGN
   | RIGHT_ASSIGN
   | AND_ASSIGN
   | XOR_ASSIGN
   | OR_ASSIGN

---

//Data Types

typed_ID
   : type_specifier ID

type
   : CHAR
   | SHORT
   | INT
   | LONG
   | FLOAT
   | DOUBLE

type_specifier //TODO Const, Volatile later
   : type
   | VOID
   | SIGNED type
   | UNSIGNED type
```

```

---

//Type List

type_specifier_list
   :  type_specifier type_specifier_list_tail

type_specifier_list_tail
   :  ε
   | ',' type_specifier_list

---

//Parameter List

parameter_specifier_list
   :  type_specifier ID parameter_specifer_tail

parameter_specifer_list_tail
   : ε
   | ','  parameter_specifier_list

---

//Program
program
   :  body EOF

---

//Body

body_statement
   : statement body_statement_tail

body_statement_tail
   : ε
   : body

body_direct_declaration
   : direct_declaration body_direct_declaration_tail

body_direct_declaration_tail
   : ε
   : body

```

```
body_function_declaration
   : function_declaration body_function_declaration_tail

body_function_declaration_tail
   : ε
   : body

body_function_prototype
   : function_declaration body_function_prototype_tail

body_function_prototype_tail
   : ε
   : body

body
   : body_statement
   | body_direct_declaration
   | body_function_declaration
   | body_function_prototype

---

//Function Declaration

function_prefix
   :  typed_ID '('

function_prototype
   :  function_prefix type_specifier_list ')' semi_colon_list
   |  function_declaration_prefix semi_colon_list

function_declaration
   :  function_prefix parameter_specifier_list ')' '{' statement '}'

function_declaration_tail
   |  semi_colon_list
   |  ε

---

//Direct Declaration

direct_declaration
   : type ID semi_colon_list
   | SIGNED type ID semi_colon_list
   | UNSIGNED type ID semi_colon_list
   | VOID ID semi_colon_list
```

```
    | CHAR ID '=' STRING_LITERAL semi_colon_list

---

//Statement and Statement List

statement
    : compound_statement //Sub statements to loops, conditional, and code blocks
    | expression_statement //Assignment, Boolean, Arithmetic Expressions
    | selection_statement //IF Statements
    | iteration_statement  //Loops
    | semi_colon_list // End of statement one or more ;
    | direct_declaration

statement_list
    : statement statement_list_tail

statement_list_tail
    : statment_list
    | ε



compound_statement
    : '{' '}'
    | '{' statement_list '}'



expression_statement
    : ';'
    | expression ';'

//TODO Add Expression
expression

selection_statement
    : IF '(' expression ')' statement
    | IF '(' expression ')' statement ELSE statement



iteration_statement
    : WHILE '(' expression ')' statement
    | FOR '(' expression_statement expression_statement ')' statement
    | FOR '(' expression_statement expression_statement expression ')' statement
```

$$program \Rightarrow body \;\; EOF \tag{1}$$

$$\begin{aligned} body \Rightarrow \;& stmt \\ \mid \;& stmt \;\; body \end{aligned} \tag{2}$$

$$\begin{aligned} stmt \Rightarrow \;& ID \;\; '=' \;\; expr \;\; ';' \\ \mid \;& IF \;\; '(' \;\; bexp \;\; ')' \;\; stmt \\ \mid \;& WHILE \;\; '(' \;\; bexp \;\; ')' \;\; stmt \\ \mid \;& '\{' \;\; substmts \\ \mid \;& ';' \end{aligned} \tag{3}$$

$$\begin{aligned} substmts \Rightarrow \;& stmt \;\; '\}' \\ \mid \;& stmt \;\; substmts \\ \mid \;& '\}' \end{aligned} \tag{4}$$

$$assignment \Rightarrow \; ID \;\; '=' \;\; expression \;\; ';' \tag{5}$$

$$\begin{aligned} expression \Rightarrow \;& expr \\ \mid \;& bexpr \end{aligned} \tag{6}$$

Notes - Continue down this route
TODO

- expr - arithmetic expressions. Make sure to get precedence (greater precedence last) and associativity correct. From `http://pages.cs.wisc.edu/~fischer/cs536.s08/course.hold/html/NOTES/3.CFG.html#assoc` Remove POW

```
exp      --> exp PLUS term        |  exp MINUS term      |  term
term     --> term TIMES factor    |  term DIVIDE factor  |  factor
```

```
factor   --> exponent POW factor  |  exponent
exponent --> MINUS exponent       |  final
final    --> INTLITERAL           |  LPAREN exp RPAREN
```

- bexpr - boolean expressions

  ```
  bexp --> TRUE
  bexp --> FALSE
  bexp --> bexp OR bexp
  bexp --> bexp AND bexp
  bexp --> NOT bexp
  bexp --> LPAREN bexp RPAREN
  ```

- stmt - add if and while loops to it. Need to make sure no ambiguity in control statements.