**Project Proposal**

The compiler is the brain that understands the language, its grammar, tokens, etc. Every language developed till now has its own specific compiler. The Lexical Analyzer can be thought of a mini compiler, that can be easily developed, modified and updated. In short, if you want to make your own rules regarding any language, then Lex Analyzer is your best friend.

# Ideas

Following are my ideas regarding developing lex analyzer:

* Experiment with different languages and their syntax
* The loops and conditional statements analyzer
* Language Identifier
* Verifying syntax of a language
* Step-wise analysis of any specific program
* Java language parser
* PHP to C Translator (Complex and just an idea till now)

I’ll concentrate to try different analyzers in order to enhance my understanding regarding lexical analyzer. The different languages syntax will be challenging but the complex programming will advance my skills as well. The source code of the analyzer will be in Python or C++ because these languages have variety of functions and I’ll be able to achieve my ideas using variant libraries offered by these languages. I personally opt PYTHON to do this project.

# Grammar

Some examples of the grammar for php to C Translator are:

* start : top\_statement\_list  
  top\_statement\_list : top\_statement\_list top\_statement

| empty

* top\_statement : statement

| function\_declaration\_statement

| HALT\_COMPILER LPAREN RPAREN SEMI\_COLON

* top\_statement : CONST constant\_declarations SEMI\_COLON
* constant\_declarations : constant\_declarations COMMA constant\_declaration

| constant\_declaration

* constant\_declaration : IDENTIFIER EQUALS static\_expr
* inner\_statement\_list : inner\_statement\_list inner\_statement

| empty

* inner\_statement : statement

| function\_declaration\_statement

| HALT\_COMPILER LPAREN RPAREN SEMI\_COLON

* inner\_statement : YIELD SEMI\_COLON

| YIELD expr SEMI\_COLON

* statement : LBRACE inner\_statement\_list RBRACE
* statement : IF LPAREN expr RPAREN statement elseif\_list else\_single

| IF LPAREN expr RPAREN COLON inner\_statement\_list new\_elseif\_list new\_else\_single ENDIF SEMI\_COLON

* statement : WHILE LPAREN expr RPAREN while\_statement
* statement : DO statement WHILE LPAREN expr RPAREN SEMI\_COLON
* statement : FOR LPAREN for\_expr SEMI\_COLON for\_expr SEMI\_COLON for\_expr RPAREN for\_statement
* statement : FOREACH LPAREN expr AS foreach\_variable foreach\_optional\_arg RPAREN foreach\_statement
* statement : SWITCH LPAREN expr RPAREN switch\_case\_list
* statement : BREAK SEMI\_COLON

| BREAK expr SEMI\_COLON

* statement : CONTINUE SEMI\_COLON

| CONTINUE expr SEMI\_COLON

* statement : RETURN SEMI\_COLON

| RETURN expr SEMI\_COLON

Some of the grammar examples for Java syntax analyzer are:

* %BasicForStatement := ~for ~( ~; ~; ~) %Statement
* %BasicForStatement := ~for ~( ~; ~; %ForUpdate ~) %Statement
* %BasicForStatement := ~for ~( ~; %Expression ~; ~) %Statement
* %BasicForStatement := ~for ~( ~; %Expression ~; %ForUpdate ~) %Statement
* %BasicForStatement := ~for ~( %ForInit ~; ~; ~) %Statement
* %BasicForStatement := ~for ~( %ForInit ~; ~; %ForUpdate ~) %Statement
* %BasicForStatement := ~for ~( %ForInit ~; %Expression ~; ~) %Statement
* %BasicForStatement := ~for ~( %ForInit ~; %Expression ~; %ForUpdate ~) %Statement

# Implementation

These are just ideas and examples about the project work. A lot of ideas can be implemented and even multiple analyzers can be summed up in this project. The real challenges will come during the coding part. I hope I’ll learn a lot through this project and try my best to accomplish a best project.