**PHP to C Translator − Lexical Analysis and Parsing**

Table of Contents

[1 Introduction 3](#_Toc71453881)

[2 Python Code 3](#_Toc71453882)

[2.1 Grammar 4](#_Toc71453883)

[2.2 Symbol Table 5](#_Toc71453884)

[2.3 Implementation 6](#_Toc71453885)

[2.3.1 Input 6](#_Toc71453886)

[2.3.2 Output 6](#_Toc71453887)

The world is moving with a rapid pace. The advancements in technology have mesmerized the man and digital media is conquering every field of this modern era. In such an advanced and digital universe, websites are an important and necessary part of the businesses, firms, hospitals, public and private organizations, government agencies, etc. **PHP** is one of the most important tools in web development. This project aims to develop a PHP to C Translator using lexical analysis and parsing.

# Overview

Lexical Analyzer is the initial part of the compiler. In this phase of the compiler development, the language’s grammar, tokens, parsing, etc. are prepared. This project is working on the php language, parsing it, tokenizing it and analysing it using Lexical analysis. This project is actually translating the front end of the php language to C language. The source code of the project is written in python language. The phenomenon of the lexical analyzer can be shown as:

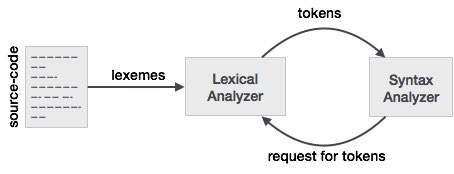


Figure : Schematic depiction of Lexical Analyzer

# Python Code

The python source code is the project’s implementation. The main part of such developments is the grammar.

## Grammar

The grammar for php tokenization and parsing is in the file “*grammar\_for\_php.txt”*. Some of the examples are given below.

|  |
| --- |
| 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 |
| statement : GLOBAL global\_var\_list SEMI\_COLON |
| statement : STATIC static\_var\_list SEMI\_COLON |
| statement : ECHO echo\_expr\_list SEMI\_COLON |
| statement : INLINE\_HTML |
| statement : expr SEMI\_COLON |
| statement : UNSET LPAREN unset\_variables RPAREN SEMI\_COLON |
| statement : SEMI\_COLON |
| {% comment %} statement : TRY LBRACE inner\_statement\_list RBRACE additional\_catches maybe\_finally |
| fully\_qualified\_class\_name : namespace\_name |
| additional\_catches : additional\_catches CATCH LPAREN fully\_qualified\_class\_name VARIABLE RPAREN LBRACE inner\_statement\_list RBRACE |
| | empty |
| maybe\_finally : FINALLY LBRACE inner\_statement\_list RBRACE |
| | empty |
| statement : THROW expr SEMI\_COLON {% endcomment %} |
| {% comment %} statement : DECLARE LPAREN declare\_list RPAREN declare\_statement # doubt |
| declare\_list : IDENTIFIER EQUALS static\_scalar |
| | declare\_list COMMA IDENTIFIER EQUALS static\_scalar # doubt |
| declare\_statement : statement |
| | COLON inner\_statement\_list ENDDECLARE SEMI\_COLON # doubt {% endcomment %} |
| elseif\_list : empty |
| | elseif\_list ELSEIF LPAREN expr RPAREN statement |
| else\_single : empty |
| | ELSE statement |

.. up to so on.

## Symbol Table

Following is the symbol table:

|  |
| --- |
| class SymbolTable:  def \_\_init\_\_(self): |
| self.symbol\_table={} |
|  |
| def free(self): |
| self.symbol\_table={} |
|  |
| def lookup(self,name): |
| return self.symbol\_table[name] |
|  |
| def insert(self,name): |
| if(self.symbol\_table.get(name,None)==None): |
| self.symbol\_table[name]={} |
| return self.symbol\_table[name] |
| return None |
| def set\_attribute(self,name,attribute\_name,attribute\_value): |
| self.symbol\_table[name][attribute\_name]=attribute\_value |
|  |
| def get\_attribute(self,name,attribute\_name): |
| return self.symbol\_table[name][attribute\_name] |

## Implementation

The *php\_lex.py* and *php\_parser.py* are the main files to run the project. A lot of php programs are available in *test* folder to test the program. Following are some examples.

### Input

<?*php*$favcolor = "red";

$x = *1*;  
  
do {  
 echo "The number is: $x <br>";  
 $x++;  
} while ($x <= *5*);

$t = date("H");  
echo "<p>The hour (of the server) is " . $t;   
echo ", and will give the following message:</p>";  
  
if ($t < "10") {  
 echo "Have a good morning!";  
} elseif ($t < "20") {  
 echo "Have a good day!";  
} else {  
 echo "Have a good night!";  
}

switch ($favcolor) {  
 case "red":  
 echo "Your favorite color is red!";  
 break;  
 case "blue":  
 echo "Your favorite color is blue!";  
 break;  
 case "green":  
 echo "Your favorite color is green!";  
 break;  
 default:  
 echo "Your favorite color is neither red, blue, nor green!";  
}  
?>

### Output

The output for the program is in the file “*parser.out*”.

Grammar  
  
Rule 0 S' -> start  
Rule 1 start -> top\_statement\_list  
Rule 2 top\_statement\_list -> top\_statement\_list top\_statement  
Rule 3 top\_statement\_list -> empty  
Rule 4 top\_statement -> statement  
Rule 5 top\_statement -> function\_declaration\_statement  
Rule 6 top\_statement -> HALT\_COMPILER LPAREN RPAREN SEMI\_COLON  
Rule 7 top\_statement -> CONST constant\_declarations SEMI\_COLON  
Rule 8 constant\_declarations -> constant\_declarations COMMA constant\_declaration  
Rule 9 constant\_declarations -> constant\_declaration  
Rule 10 constant\_declaration -> IDENTIFIER EQUALS static\_expr  
Rule 11 inner\_statement\_list -> inner\_statement\_list inner\_statement  
Rule 12 inner\_statement\_list -> empty  
Rule 13 inner\_statement -> statement  
Rule 14 inner\_statement -> function\_declaration\_statement  
Rule 15 inner\_statement -> YIELD SEMI\_COLON  
Rule 16 inner\_statement -> YIELD expr SEMI\_COLON  
Rule 17 statement -> LBRACE inner\_statement\_list RBRACE  
Rule 18 statement -> IF LPAREN expr RPAREN statement elseif\_list  
Rule 19 statement -> IF LPAREN expr RPAREN statement elseif\_list ELSE statement

(244) static\_expr -> static\_expr . AND static\_expr  
(245) static\_expr -> static\_expr . OR static\_expr  
(246) static\_expr -> static\_expr . XOR static\_expr  
(247) static\_expr -> static\_expr . CONCAT static\_expr  
(248) static\_expr -> static\_expr . PLUS static\_expr  
(249) static\_expr -> static\_expr . MINUS static\_expr  
(250) static\_expr -> static\_expr . MUL static\_expr  
(251) static\_expr -> static\_expr . DIV static\_expr  
(252) static\_expr -> static\_expr . SL static\_expr  
(253) static\_expr -> static\_expr . SR static\_expr  
(254) static\_expr -> static\_expr . MOD static\_expr  
(255) static\_expr -> static\_expr . IS\_IDENTICAL static\_expr  
(256) static\_expr -> static\_expr . IS\_NOT\_IDENTICAL static\_expr  
(257) static\_expr -> static\_expr . IS\_EQUAL\_TO static\_expr  
(258) static\_expr -> static\_expr . IS\_NOT\_EQUAL static\_expr  
(259) static\_expr -> static\_expr . LESS\_THAN static\_expr  
(260) static\_expr -> static\_expr . LESS\_THAN\_OR\_EQUAL static\_expr

.. up to so on.

# Conclusion

The project is analyzing and parsing the php programs accurately. The output is detailed with rules, states and step-by-step explanation at each state. The advanced php programs will require more grammar and work but with intermediate programs, this project is working as desired.