```
# A simple lexical analyzer for PHP source code using PLY

# Sample code
# Example PHP code snippets to analyze

# Example 1: Simple Print Statements
code_1 = '''
<?php //php 7.2.24
    PRINT "Hello, world! \n";
    ECHO "Welcome "
?>
'''

# Example 2: Simple Arithmetic Expression
code_2 = '''
<?php
    $num1 = 10;
    $num2 = 20;
    $num3 = 30;
    $sum = $num1 + $num2 + $num3;
    $avg = $sum/3;
    PRINT "Num1 is " . $num1 ."\n";
    PRINT "Num2 is " . $num2 ."\n";
    PRINT "Num3 is " . $num3 ."\n";
    PRINT "Sum 3 numbers is " . $sum ."\n";
    PRINT "Average is " . $avg;
?>
'''

# Example 3: Simple Conditional IF with Block
code_3 = '''
<?php //php 7.2.24
    $num1 = 10;
    $num2 = 20;
    IF ($num1 > $num2) {
        $bignum = $num1;
        PRINT "Big Number is " . $bignum;
    }
    ELSE {
        $bignum = $num2;
        PRINT "Big Number is " . $bignum;
    }
?>
'''

# Example 4: Simple Looping with Conditional IF Block
code_4 = '''
<?php //php 7.2.24
    PRINT "List of Odd Number 1-100:\n";
    PRINT "\n";
    $num = 1;
    WHILE ($num <= 100) {
        IF (($num % 2) != 0) {
            $oddnum=$num
            PRINT "" . $num . " "; }
        $num=$num + 1;
    }
?>
'''


#import lex from python lex yacc
from ply import lex

# List of PHP keywords
keywords = [
```

```
    'PRINT',
    'ECHO',
    'IF',
    'ELSE',
    'WHILE',
    'RETURN',
]

# List of tokens
tokens = keywords + [
    'EQUALS',
    'PLUS',
    'DIVIDE',
    'LPAREN',
    'RPAREN',
    'RCURLY',
    'LCURLY',
    'LESSEQUAL',
    'GREATERTHAN',
    'CONCAT',
    'SEMI',
    'NOT',
    'MOD',
    'IDENTIFIER',
    'NUMBER',
    'STRING',
]

# Ignore whitespace
t_ignore = ' \t'

# Ignore PHP tags
def t_ignore_PHP_OPEN(t):
    r'\<\?php'
    pass

def t_ignore_PHP_CLOSE(t):
    r'\?\>'
    pass

# Ignore comment
t_ignore_COMMENT =r'//.*'

# Token definitions
def t_KEYWORD(t):
    r'[a-zA-Z_]+'
    if t.value in keywords:
        t.type = t.value
    return t

def t_IDENTIFIER(t):
    r'\$[a-zA-Z_][a-zA-Z0-9_]*'  # Identifiers start with $ followed by letters or numbers
    return t

def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)  # Convert to integer
    return t

def t_STRING(t):
    r'"([^"\\]*(\\.[^"\\]*)*)"'
    return t

t_EQUALS = r'\='
t_PLUS = r'\+'
t_DIVIDE = r'\/'
```

```
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LCURLY = r'\{'
t_RCURLY = r'\}'
t_LESSEQUAL = r'\<='
t_GREATERTHAN = r'\>'
t_CONCAT = r'\.'  # Concatenation operator
t_SEMI = r'\;'
t_NOT = r'\!'  # Not operator
t_MOD = r'\%'  # Modulo operator

def t_NEWLINE(t):
    r'\n'
    t.lexer.lineno += len(t.value)

def t_error(t):
    print(f"Illegal character '{t.value[0]}' at line {t.lineno}")
    t.lexer.skip(1)

# Build the lexer
lex.lex(debug=0)

# Function to analyze input code
def analyze_code(code):
    lex.input(code)
    while True:
        token = lex.token()  # Get the next token
        if not token:  # If there are no more tokens, break
            break
        print(token)

# Analyze each example PHP code
print("Analyzing PHP Code 1:")
analyze_code(code_1)
print("\nAnalyzing PHP Code 2:")
analyze_code(code_2)
print("\nAnalyzing PHP Code 3:")
analyze_code(code_3)
print("\nAnalyzing PHP Code 4:")
analyze_code(code_4)
```