

# Assignment 2

Assignment Syntax Validation of a programming language by writing the Context Free Grammar. (PLY Tools)

|                                |                             |                        |
|--------------------------------|-----------------------------|------------------------|
| <b>Batch No.</b>               | <b>2025</b>                 | <b>Date:11/11/2025</b> |
| <b>Name 1: Ibrahim khaleel</b> | <b>SRN 1: PES2UG25CS809</b> |                        |
| <b>Name 2:Dheeraj</b>          | <b>SRN 2: PES2UG25CS806</b> |                        |
| <b>Programming Language:</b>   |                             |                        |
| <b>List of constructs:</b>     |                             |                        |
| <b>Programming Language:</b>   |                             |                        |
| <b>C++</b>                     |                             |                        |
| <b>List of constructs:</b>     |                             |                        |
| <b>1. Do-While Loop</b>        |                             |                        |
| <b>2. Nested If-Else</b>       |                             |                        |
| <b>3. For Loop</b>             |                             |                        |
| <b>4. Switch-Case</b>          |                             |                        |
| <b>5. Function Definition</b>  |                             |                        |

## Lex Program:

```
lex.py
lex.py
1  import ply.lex as lex
2
3  # List of token names
4  # Add COMMA to tokens list
5  tokens = (
6      'DO', 'WHILE', 'FOR', 'IF', 'ELSE', 'SWITCH', 'CASE', 'DEFAULT', 'BREAK',
7      'RETURN', 'INT', 'VOID', 'IDENTIFIER', 'NUMBER',
8      'LPAREN', 'RPAREN', 'LBRACE', 'RBRACE', 'SEMI', 'COLON', 'COMMA',
9      'PLUS', 'MINUS', 'MULT', 'DIV', 'ASSIGN',
10     'EQ', 'NE', 'LT', 'LE', 'GT', 'GE'
11 )
12
13 # Add comma rule
14 t_COMMA = r','
15
16 # Regular expression rules for simple tokens
17 t_DO = r'do'
18 t_WHILE = r'while'
19 t_FOR = r'for'
20 t_IF = r'if'
21 t_ELSE = r'else'
22 t_SWITCH = r'switch'
23 t_CASE = r'case'
24 t_DEFAULT = r'default'
25 t_BREAK = r'break'
26 t_RETURN = r'return'
27 t_INT = r'int'
28 t_VOID = r'void'
29 t_LPAREN = r'\('
30 t_RPAREN = r'\)'
31 t_LBRACE = r'\{'
32 t_RBRACE = r'\}'
33 t_SEMI = r';'
34 t_COLON = r':'
35 t_PLUS = r'\+'
36 t_MINUS = r'\-'
37 t_MULT = r'\*'
38 t_DIV = r'/'
39 t_ASSIGN = r'='
40 t_EQ = r'=='
41 t_NE = r'!='
42 t_LT = r'<'
43 t_LE = r'<='
44 t_GT = r'>'
45 t_GE = r'>='
46
47 # A regular expression rule for identifiers
```

nd; maybe you need to run "Codeium Auth?"

```

lex.py
lex.py
44 t_GE = r'>='
45
46
47 # A regular expression rule for identifiers
48 def t_IDENTIFIER(t):
49     r'[a-zA-Z_][a-zA-Z_0-9]*'
50     # Check if the identifier is a reserved keyword
51     reserved = {}
52     reserved['do'] = 'DO',
53     reserved['while'] = 'WHILE',
54     reserved['for'] = 'FOR',
55     reserved['if'] = 'IF',
56     reserved['else'] = 'ELSE',
57     reserved['switch'] = 'SWITCH',
58     reserved['case'] = 'CASE',
59     reserved['default'] = 'DEFAULT',
60     reserved['break'] = 'BREAK',
61     reserved['return'] = 'RETURN',
62     reserved['int'] = 'INT',
63     reserved['void'] = 'VOID'
64
65     t.type = reserved.get(t.value, 'IDENTIFIER')
66     return t
67
68 # A regular expression rule for numbers
69 def t_NUMBER(t):
70     r'\d+'
71     t.value = int(t.value)
72     return t
73
74 # Define a rule to track line numbers (useful for error messages)
75 def t_newline(t):
76     r'\n+'
77     t.lexer.lineno += len(t.value)
78
79 # A string containing ignored characters (spaces and tabs)
80 t_ignore = ' \t'
81
82 # Error handling rule
83 def t_error(t):
84     print(f'Illegal character '{t.value[0]}' at line {t.lineno}')
85     t.lexer.skip(1)
86
87 # Build the lexer
88 lexer = lex.lex()
89
90 # Test data (optional, for individual testing)
91 if __name__ == '__main__':
92     data = """
93     int main() {
94         int x = 5;
95         if (x > 0) {
96             return 1;
97         } else {
98             return 0;
99         }
100     }
101     """
102     lexer.input(data)
103     for tok in lexer:
104         print(tok)
105

```

## Yacc Program:

```
W Go Run Terminal Help
yacc.py x
yacc.py
1 import ply.yacc as yacc
2 from lex import tokens, lexer
3
4 # --- Define the grammar rules ---
5
6 # Starting point of the grammar
7 def p_program(p):
8     '''program : function_definition'''
9     print("Syntax is valid!")
10
11 # 1. Function Definition Construct
12 def p_function_definition(p):
13     '''function_definition : type IDENTIFIER LPAREN parameters RPAREN LBRACE statements RBRACE'''
14
15 def p_parameters(p):
16     '''parameters : parameter_list
17     | empty'''
18
19 def p_parameter_list(p):
20     '''parameter_list : parameter_list COMMA parameter
21     | parameter'''
22
23 def p_parameter(p):
24     '''parameter : type IDENTIFIER'''
25
26 def p_type(p):
27     '''type : INT
28     | VOID'''
29
30 # 2. For Loop Construct
31 def p_for_loop(p):
32     '''for_loop : FOR LPAREN for_init SEMI condition SEMI for_update RPAREN LBRACE statements RBRACE'''
33
34 def p_for_init(p):
35     '''for_init : assignment
36     | declaration
37     | empty'''
38
39 def p_for_update(p):
40     '''for_update : assignment
41     | empty'''
42
43 # 3. Do-While Loop Construct
44 def p_do_while_loop(p):
45     '''do_while_loop : DO LBRACE statements RBRACE WHILE LPAREN condition RPAREN SEMI'''
46
47 # 4. Nested If-Else Construct
48 def p_if_else_statement(p):
49     '''if_else_statement : IF LPAREN condition RPAREN LBRACE statements RBRACE else_part'''
50
51 def p_else_part(p):
52     '''else_part : ELSE LBRACE statements RBRACE
53     | ELSE if_else_statement
54     | empty'''
55
56 # 5. Switch-Case Construct
57 def p_switch_statement(p):
58
59 and, maybe you need to run `Codeium Auth?
```

```

View Go Run Terminal Help
yacc.py x
yacc.py
55
56 # 5. Switch-Case Construct
57 def p_switch_statement(p):
58     '''switch_statement : SWITCH LPAREN expression RPAREN LBRACE case_list RBRACE'''
59
60 def p_case_list(p):
61     '''case_list : case_list case
62     | case'''
63
64 def p_case(p):
65     '''case : CASE expression COLON statements BREAK SEMI
66     | DEFAULT COLON statements BREAK SEMI'''
67
68 # --- Supporting Grammar Rules ---
69
70 def p_statements(p):
71     '''statements : statements statement
72     | statement
73     | empty'''
74
75 def p_statement(p):
76     '''statement : expression_statement
77     | declaration_statement
78     | if_else_statement
79     | for_loop
80     | do_while_loop
81     | switch_statement
82     | return_statement
83     | LBRACE statements RBRACE'''
84
85 def p_expression_statement(p):
86     '''expression_statement : expression SEMI
87     | assignment SEMI
88     | SEMI''' # empty statement
89
90 def p_declaration_statement(p):
91     '''declaration_statement : declaration SEMI'''
92
93 def p_declaration(p):
94     '''declaration : type IDENTIFIER
95     | type IDENTIFIER ASSIGN expression'''
96
97 def p_assignment(p):
98     '''assignment : IDENTIFIER ASSIGN expression'''
99
100 def p_condition(p):
101     '''condition : expression'''
102
103 def p_expression(p):
104     '''expression : simple_expression
105     | simple_expression relation simple_expression'''
106
107 def p_simple_expression(p):
108     '''simple_expression : term
109     | simple_expression PLUS term
110     | simple_expression MINUS term'''
111
found; maybe you need to run 'Codeium Auth'?

```

```

ew Go Run Terminal Help
yacc.py x
yacc.py
103 def p_expression(p):
104     """expression : simple_expression
106
107 def p_simple_expression(p):
108     """simple_expression : term
109     | simple_expression PLUS term
110     | simple_expression MINUS term"""
111
112 def p_term(p):
113     """term : factor
114     | term MULT factor
115     | term DIV factor"""
116
117 def p_factor(p):
118     """factor : IDENTIFIER
119     | NUMBER
120     | LPAREN expression RPAREN"""
121
122 def p_relation(p):
123     """relation : EQ
124     | NE
125     | LT
126     | LE
127     | GT
128     | GE"""
129
130 def p_return_statement(p):
131     """return_statement : RETURN expression SEMI
132     | RETURN SEMI"""
133
134 def p_empty(p):
135     """empty :
136     pass
137
138 # --- Error Handling ---
139 def p_error(p):
140     if p:
141         print(f"Syntax error at token '{p.value}' (type: {p.type}) at line {p.lineno}")
142         # Skip ahead to the next semicolon or brace to continue parsing
143         parser.errorok()
144         # Look for next semicolon, brace, or newline to resync
145         while True:
146             tok = parser.token()
147             if not tok or tok.type in ['SEMI', 'LBRACE', 'RBRACE']:
148                 break
149         else:
150             print("Syntax error at EOF")
151
152 # Build the parser
153 parser = yacc.yacc()
154
155 # --- Test the parser with the chosen constructs ---
156 if __name__ == '__main__':
157     print("=" * 50)
158     print("TESTING C++ CONSTRUCTS WITH PLY")
159     print("=" * 50)
160
und, maybe you need to run `Codeium Auth?

```

```
Go Run Terminal Help
yacc.py x
yacc.py
39 def p_error(p):
40     else:
41         print("Syntax error at EOF")
42
43 # Build the parser
44 parser = yacc.yacc()
45
46 # --- Test the parser with the chosen constructs ---
47 if __name__ == '__main__':
48     print("=" * 50)
49     print("TESTING C++ CONSTRUCTS WITH PLY")
50     print("=" * 50)
51
52     test_cases = [
53         # Test 1: Do-While Loop
54         """
55         int main() {
56             do {
57                 x = x - 1;
58             } while (x > 0);
59         }
60         """,
61         # Test 2: Nested If-Else
62         """
63         void check() {
64             if (a == 10) {
65                 if (b < a) {
66                     result = 1;
67                 } else {
68                     result = 2;
69                 }
70             } else {
71                 result = 3;
72             }
73         }
74         """,
75         # Test 3: For Loop
76         """
77         int loop() {
78             for (i = 0; i < 10; i = i + 1) {
79                 sum = sum + i;
80             }
81         }
82         """,
83         # Test 4: Switch-Case
84         """
85         void choice() {
86             switch (option) {
87                 case 1:
88                     x = 10;
89                     break;
90                 case 2:
91                     x = 20;
92                     break;
93                 default:
94                     x = 0;
95                     break;
96             }
97         }
98         """
99     ]
100
101     for test_case in test_cases:
102         parser.parse(test_case, debug=False)
```

maybe you need to run `Codeium Auth`

## Screen Shot of Output

```
ibrahimfadu .../Cls_mini/AFLFLL main ! 21:20 python yacc.py
```

```
=====
TESTING C++ CONSTRUCTS WITH PLY
=====
```

```
--- Test Case 1: ---
```

```
Input Code Snippet:
```

```
int main() {
    do {
        x = x - 1;
    } while (x > 0);
}
```

```
Parser Output:
```

```
Syntax is valid!
```

```
✅ SUCCESS: Parsing completed without syntax errors.
```

```
--- Test Case 2: ---
```

```
Input Code Snippet:
```

```
void check() {
    if (a == 10) {
        if (b < a) {
            result = 1;
        } else {
            result = 2;
        }
    } else {
        result = 3;
    }
}
```

```
Parser Output:
```

```
Syntax is valid!
```

```
✅ SUCCESS: Parsing completed without syntax errors.
```

```
--- Test Case 3: ---
```

```
Input Code Snippet:
```

```
int loop() {
    for (i = 0; i < 10; i = i + 1) {
        sum = sum + i;
    }
}
```

```
Parser Output:
```

```
Syntax is valid!
```

```
✅ SUCCESS: Parsing completed without syntax errors.
```

```
--- Test Case 4: ---
```

```
Input Code Snippet:
```

```
void choice() {
    switch (option) {
        case 1:
            x = 10;
            break;
        case 2:
            x = 20;
            break;
        default:
            x = 0;
    }
}
```



--- Test Case 3: ---

Input Code Snippet:

```
int loop() {  
    for (i = 0; i < 10; i = i + 1) {  
        sum = sum + i;  
    }  
}
```

Parser Output:

Syntax is valid!

✅ SUCCESS: Parsing completed without syntax errors.

--- Test Case 4: ---

Input Code Snippet:

```
void choice() {  
    switch (option) {  
        case 1:  
            x = 10;  
            break;  
        case 2:  
            x = 20;  
            break;  
        default:  
            x = 0;  
            break;  
    }  
}
```

Parser Output:

Syntax is valid!

✅ SUCCESS: Parsing completed without syntax errors.

--- Test Case 5: ---

Input Code Snippet:

```
int main() {  
    int i = 0;  
    for (i = 0; i < 5; i = i + 1) {  
        if (i == 1) {  
            x = 10;  
        } else {  
            switch (x) {  
                case 10:  
                    break;  
                default:  
                    break;  
            }  
        }  
    }  
    do {  
        i = i - 1;  
    } while (i > 0);  
    return 0;  
}
```

Parser Output:

Syntax is valid!

✅ SUCCESS: Parsing completed without syntax errors.

ibrahimfadu .../Cls\_mini/AFLF main ! 21:20 ;9~|