Assignment 2

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

Batch No.	2025	Date:11/11/2025
Name 1: Ibrahim khaleel	SRN 1: PES2UG25CS809	
Name 2:Dheeraj	SRN 2: PES2UG25CS806	

Programming Language:

List of constructs:

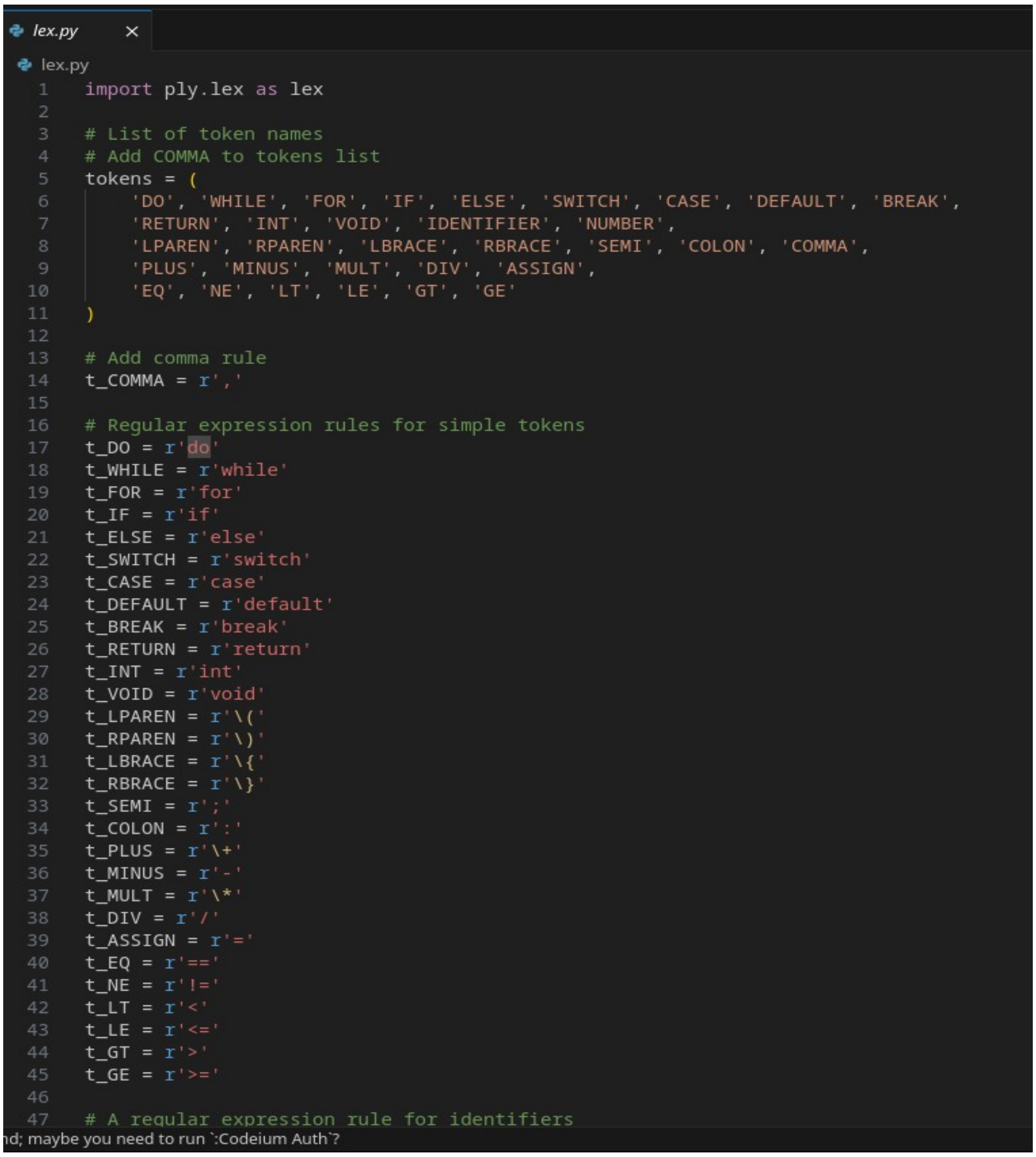
Programming Language:

C++

List of constructs:

1. **Do-While Loop**
2. **Nested If-Else**
3. **For Loop**
4. **Switch-Case**
5. **Function Definition**

Lex Program:

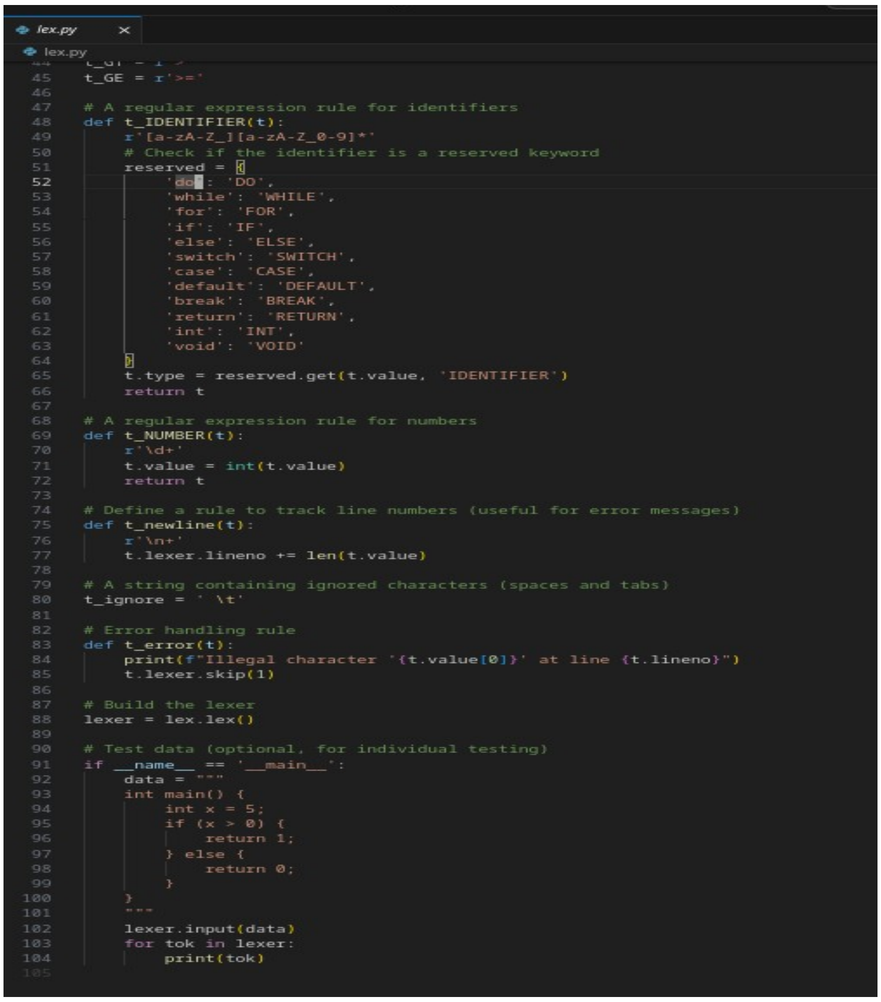


lex.py

lex.py

```
import ply.lex as lex

# List of token names
# Add COMMA to tokens list
tokens = (
    'DO', 'WHILE', 'FOR', 'IF', 'ELSE', 'SWITCH', 'CASE', 'DEFAULT', 'BREAK',
    'RETURN', 'INT', 'VOID', 'IDENTIFIER', 'NUMBER',
    'LPAREN', 'RPAREN', 'LBRACE', 'RBRACE', 'SEMI', 'COLON', 'COMMA',
    'PLUS', 'MINUS', 'MULT', 'DIV', 'ASSIGN',
    'EQ', 'NE', 'LT', 'LE', 'GT', 'GE'
)

# Add comma rule
t_COMMA = r','

# Regular expression rules for simple tokens
t_DO = r'do'
t_WHILE = r'while'
t_FOR = r'for'
t_IF = r'if'
t_ELSE = r'else'
t_SWITCH = r'switch'
t_CASE = r'case'
t_DEFAULT = r'default'
t_BREAK = r'break'
t_RETURN = r'return'
t_INT = r'int'
t_VOID = r'void'
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_LBRACE = r'\{'
t_RBRACE = r'\}'
t_SEMI = r';'
t_COLON = r':'
t_PLUS = r'\+'
t_MINUS = r'-'
t_MULT = r'\*'
t_DIV = r'/'
t_ASSIGN = r'='
t_EQ = r'=='
t_NE = r'!='
t_LT = r'<'
t_LE = r'<='
t_GT = r'>'
t_GE = r'>='

# A regular expression rule for identifiers
```

nd; maybe you need to run `:Codeium Auth`?

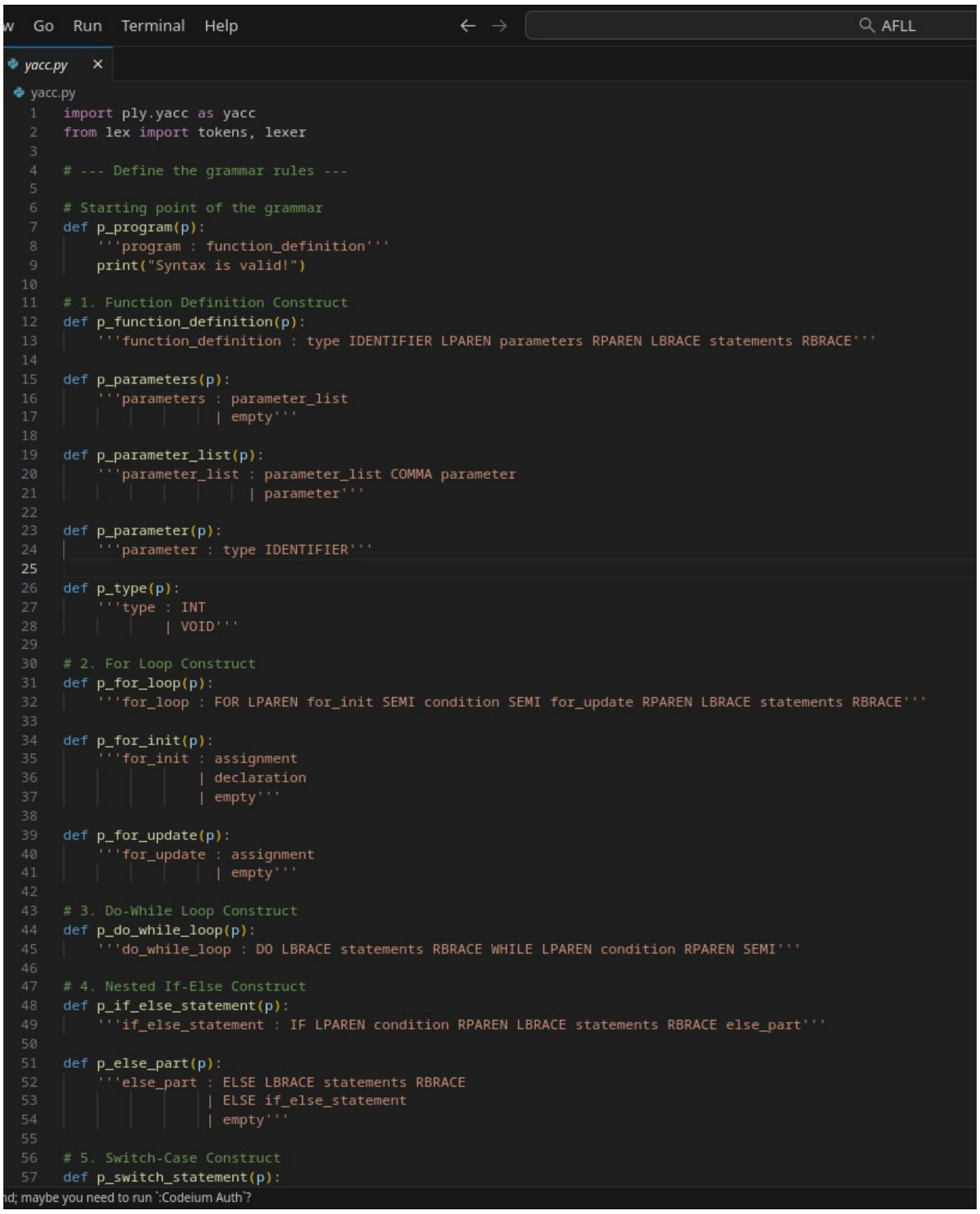


lex.py

lex.py

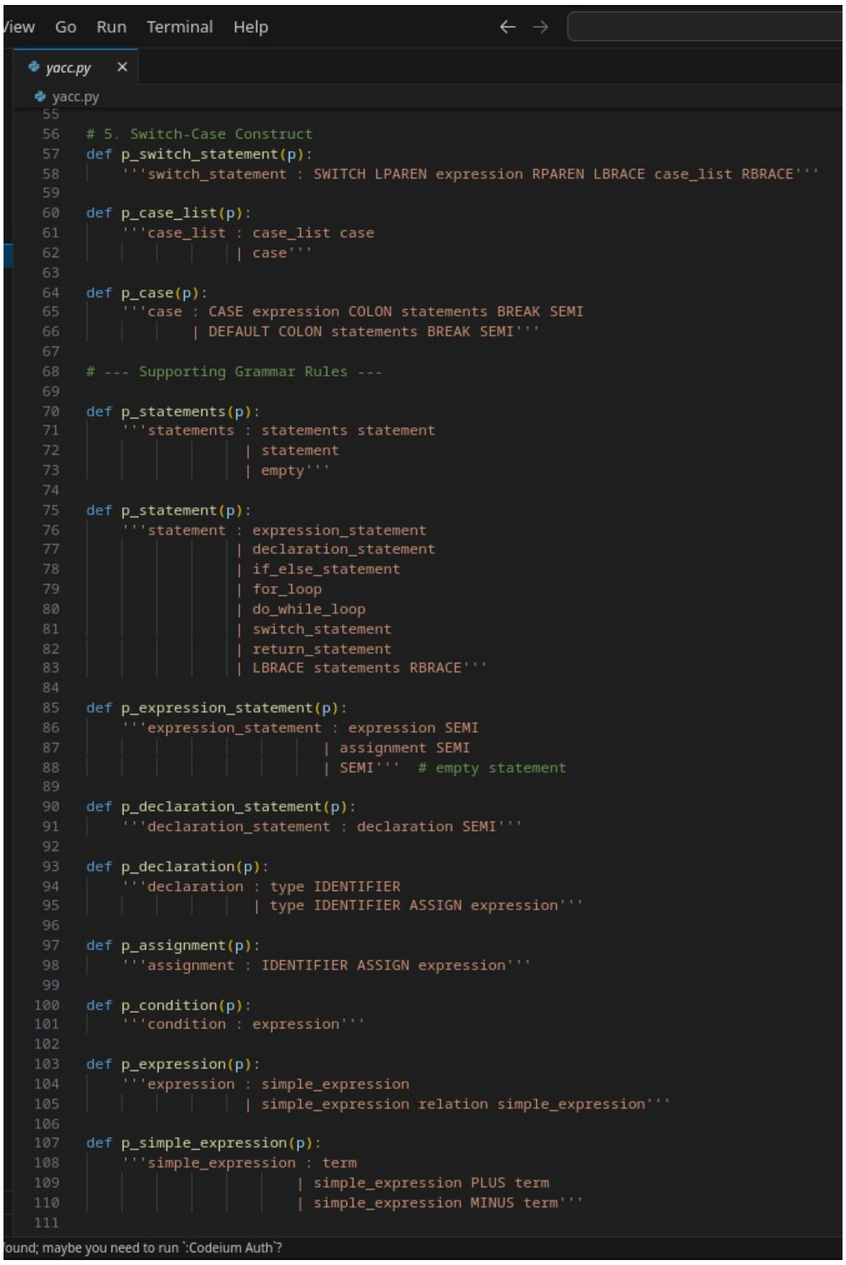
```python
t_GE = r'>='

# A regular expression rule for identifiers
def t_IDENTIFIER(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    # Check if the identifier is a reserved keyword
    reserved = {
        'do': 'DO',
        'while': 'WHILE',
        'for': 'FOR',
        'if': 'IF',
        'else': 'ELSE',
        'switch': 'SWITCH',
        'case': 'CASE',
        'default': 'DEFAULT',
        'break': 'BREAK',
        'return': 'RETURN',
        'int': 'INT',
        'void': 'VOID'
    }
    t.type = reserved.get(t.value, 'IDENTIFIER')
    return t

# A regular expression rule for numbers
def t_NUMBER(t):
    r'\d+'
    t.value = int(t.value)
    return t

# Define a rule to track line numbers (useful for error messages)
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# A string containing ignored characters (spaces and tabs)
t_ignore = ' \t'

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

# Build the lexer
lexer = lex.lex()

# Test data (optional, for individual testing)
if __name__ == '__main__':
    data = """
    int main() {
        int x = 5;
        if (x > 0) {
            return 1;
        } else {
            return 0;
        }
    }
    """
    lexer.input(data)
    for tok in lexer:
        print(tok)
```

Yacc Program:



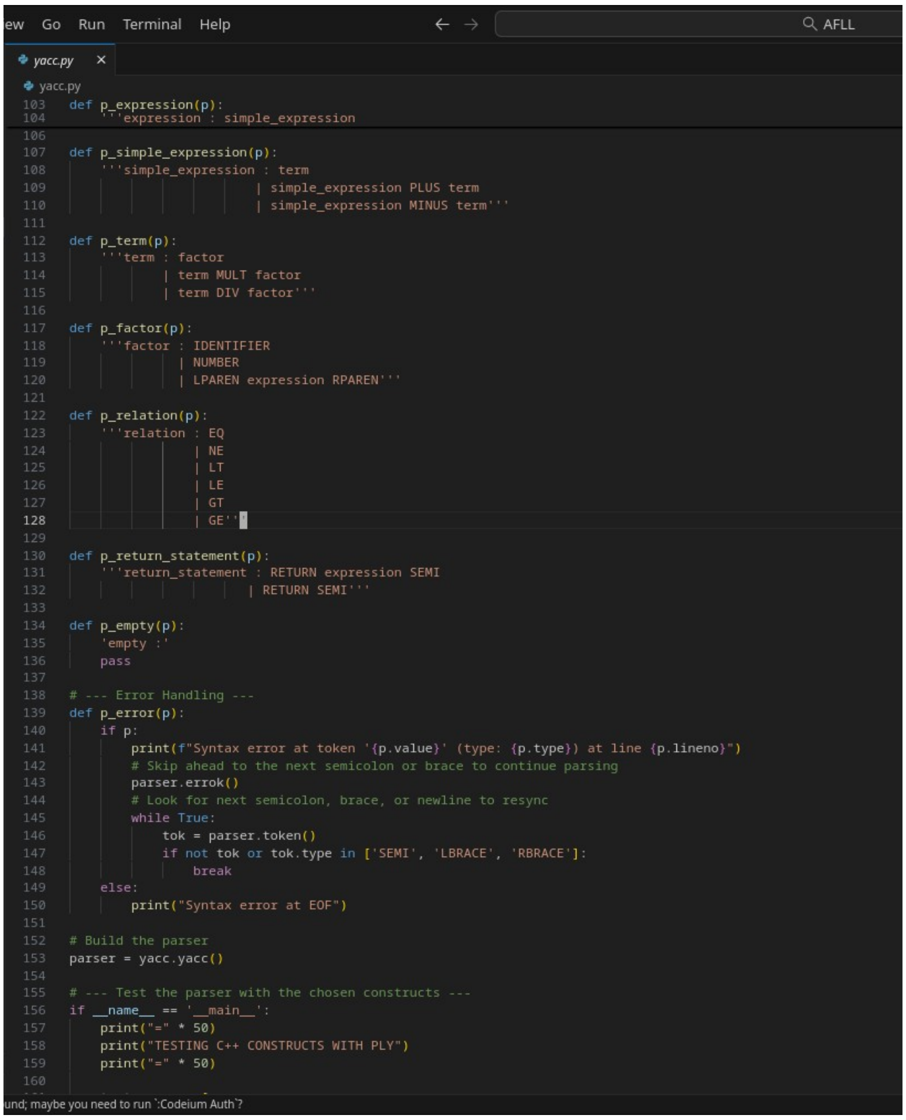
```python
import ply.yacc as yacc
from lex import tokens, lexer

# --- Define the grammar rules ---

# Starting point of the grammar
def p_program(p):
    '''program : function_definition'''
    print("Syntax is valid!")

# 1. Function Definition Construct
def p_function_definition(p):
    '''function_definition : type IDENTIFIER LPAREN parameters RPAREN LBRACE statements RBRACE'''

def p_parameters(p):
    '''parameters : parameter_list
                  | empty'''

def p_parameter_list(p):
    '''parameter_list : parameter_list COMMA parameter
                      | parameter'''

def p_parameter(p):
    '''parameter : type IDENTIFIER'''

def p_type(p):
    '''type : INT
            | VOID'''

# 2. For Loop Construct
def p_for_loop(p):
    '''for_loop : FOR LPAREN for_init SEMI condition SEMI for_update RPAREN LBRACE statements RBRACE'''

def p_for_init(p):
    '''for_init : assignment
                | declaration
                | empty'''

def p_for_update(p):
    '''for_update : assignment
                  | empty'''

# 3. Do-While Loop Construct
def p_do_while_loop(p):
    '''do_while_loop : DO LBRACE statements RBRACE WHILE LPAREN condition RPAREN SEMI'''

# 4. Nested If-Else Construct
def p_if_else_statement(p):
    '''if_else_statement : IF LPAREN condition RPAREN LBRACE statements RBRACE else_part'''

def p_else_part(p):
    '''else_part : ELSE LBRACE statements RBRACE
                 | ELSE if_else_statement
                 | empty'''

# 5. Switch-Case Construct
def p_switch_statement(p):
```

nd; maybe you need to run `:Codeium Auth`?



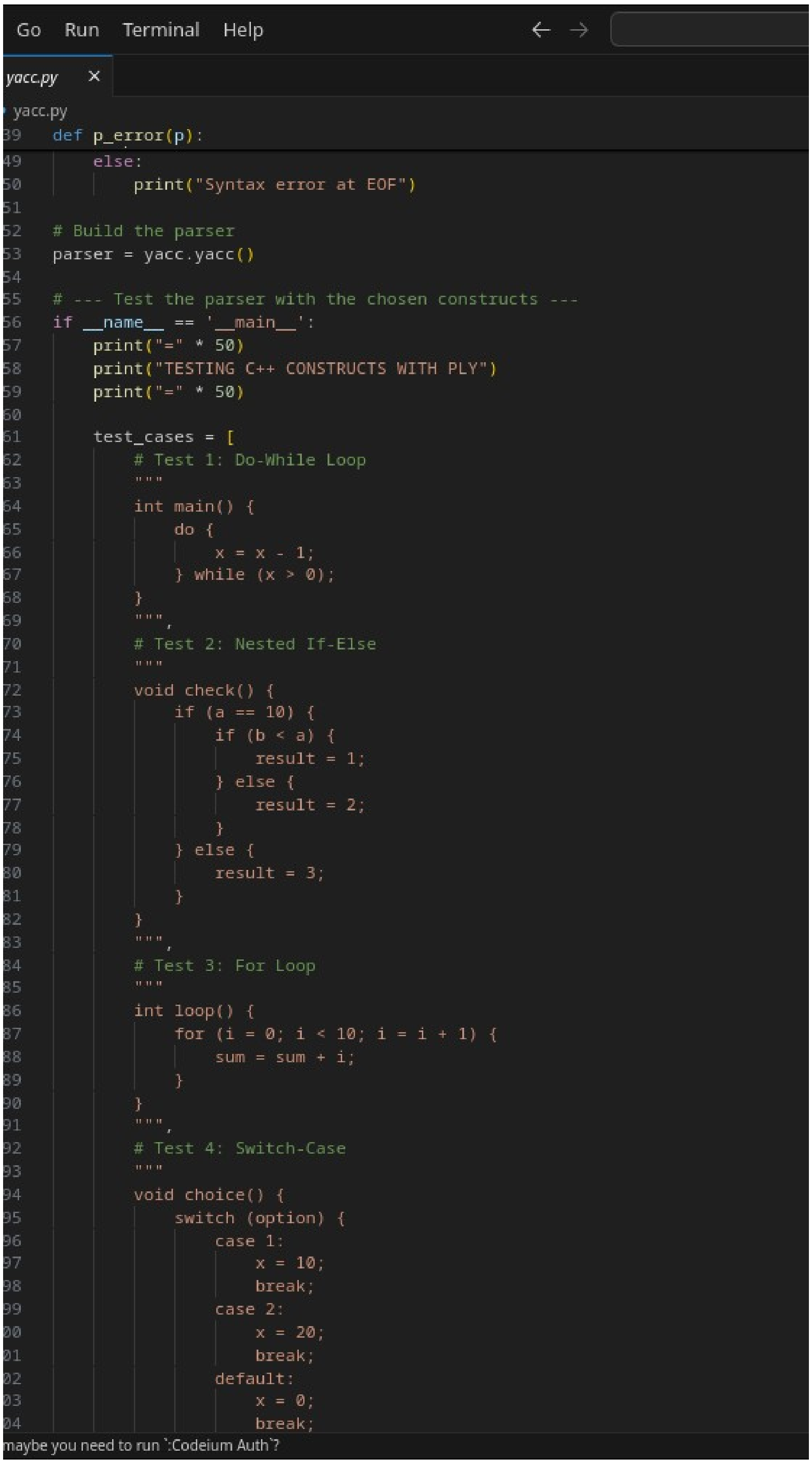
```python

# 5. Switch-Case Construct
def p_switch_statement(p):
    '''switch_statement : SWITCH LPAREN expression RPAREN LBRACE case_list RBRACE'''

def p_case_list(p):
    '''case_list : case_list case
                 | case'''

def p_case(p):
    '''case : CASE expression COLON statements BREAK SEMI
            | DEFAULT COLON statements BREAK SEMI'''

# --- Supporting Grammar Rules ---

def p_statements(p):
    '''statements : statements statement
                  | statement
                  | empty'''

def p_statement(p):
    '''statement : expression_statement
                 | declaration_statement
                 | if_else_statement
                 | for_loop
                 | do_while_loop
                 | switch_statement
                 | return_statement
                 | LBRACE statements RBRACE'''

def p_expression_statement(p):
    '''expression_statement : expression SEMI
                          | assignment SEMI
                          | SEMI'''  # empty statement

def p_declaration_statement(p):
    '''declaration_statement : declaration SEMI'''

def p_declaration(p):
    '''declaration : type IDENTIFIER
                   | type IDENTIFIER ASSIGN expression'''

def p_assignment(p):
    '''assignment : IDENTIFIER ASSIGN expression'''

def p_condition(p):
    '''condition : expression'''

def p_expression(p):
    '''expression : simple_expression
                  | simple_expression relation simple_expression'''

def p_simple_expression(p):
    '''simple_expression : term
                        | simple_expression PLUS term
                        | simple_expression MINUS term'''

```

ound; maybe you need to run `:Codeium Auth`?



```
def p_expression(p):
    '''expression : simple_expression

def p_simple_expression(p):
    '''simple_expression : term
                        | simple_expression PLUS term
                        | simple_expression MINUS term'''

def p_term(p):
    '''term : factor
            | term MULT factor
            | term DIV factor'''

def p_factor(p):
    '''factor : IDENTIFIER
              | NUMBER
              | LPAREN expression RPAREN'''

def p_relation(p):
    '''relation : EQ
                | NE
                | LT
                | LE
                | GT
                | GE'''

def p_return_statement(p):
    '''return_statement : RETURN expression SEMI
                       | RETURN SEMI'''

def p_empty(p):
    'empty :'
    pass

# --- Error Handling ---
def p_error(p):
    if p:
        print(f"Syntax error at token '{p.value}' (type: {p.type}) at line {p.lineno}")
        # Skip ahead to the next semicolon or brace to continue parsing
        parser.errok()
        # Look for next semicolon, brace, or newline to resync
        while True:
            tok = parser.token()
            if not tok or tok.type in ['SEMI', 'LBRACE', 'RBRACE']:
                break
    else:
        print("Syntax error at EOF")

# Build the parser
parser = yacc.yacc()

# --- Test the parser with the chosen constructs ---
if __name__ == '__main__':
    print("=" * 50)
    print("TESTING C++ CONSTRUCTS WITH PLY")
    print("=" * 50)

```

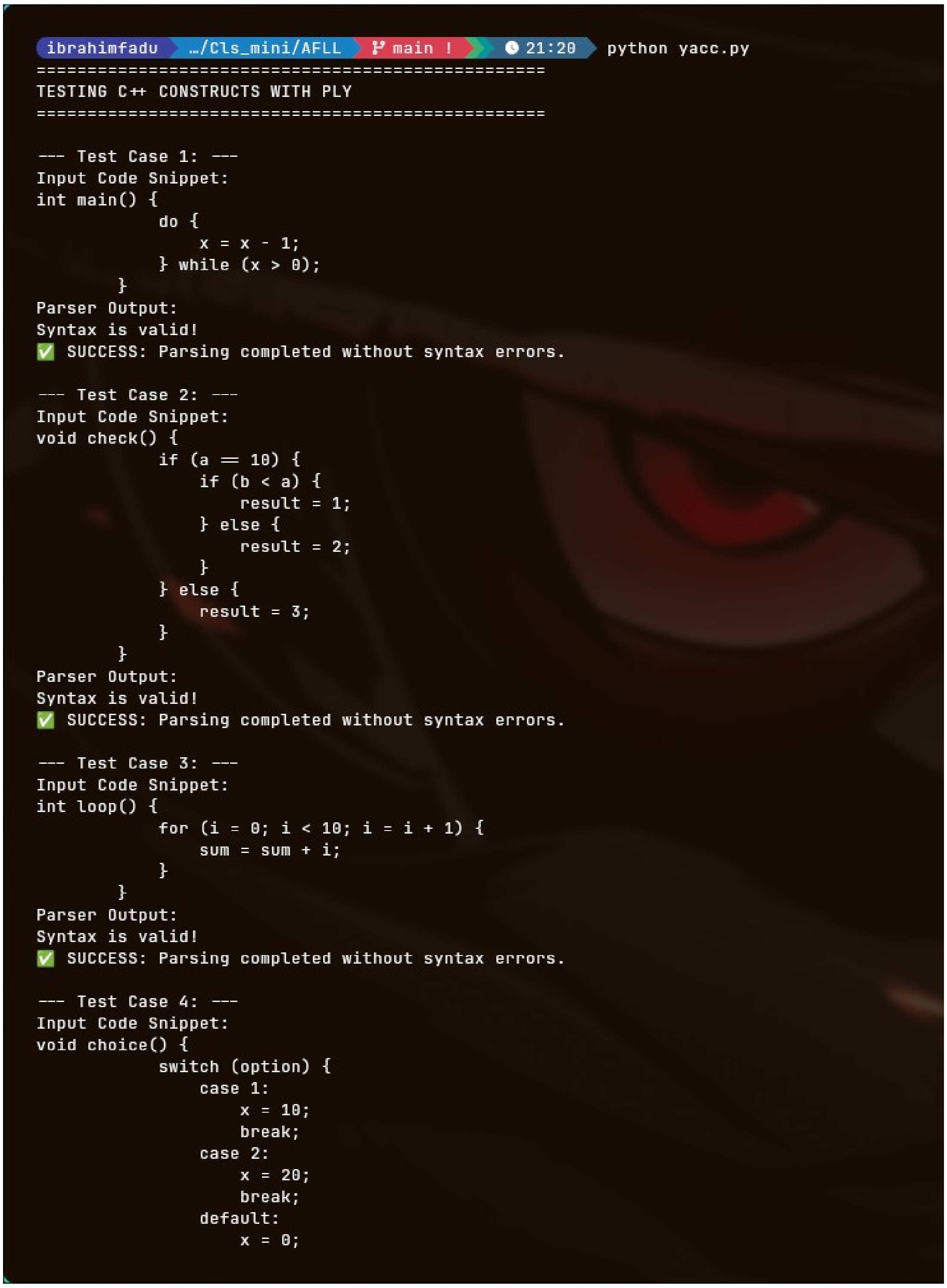
```
und; maybe you need to run `:Codeium Auth`?
```



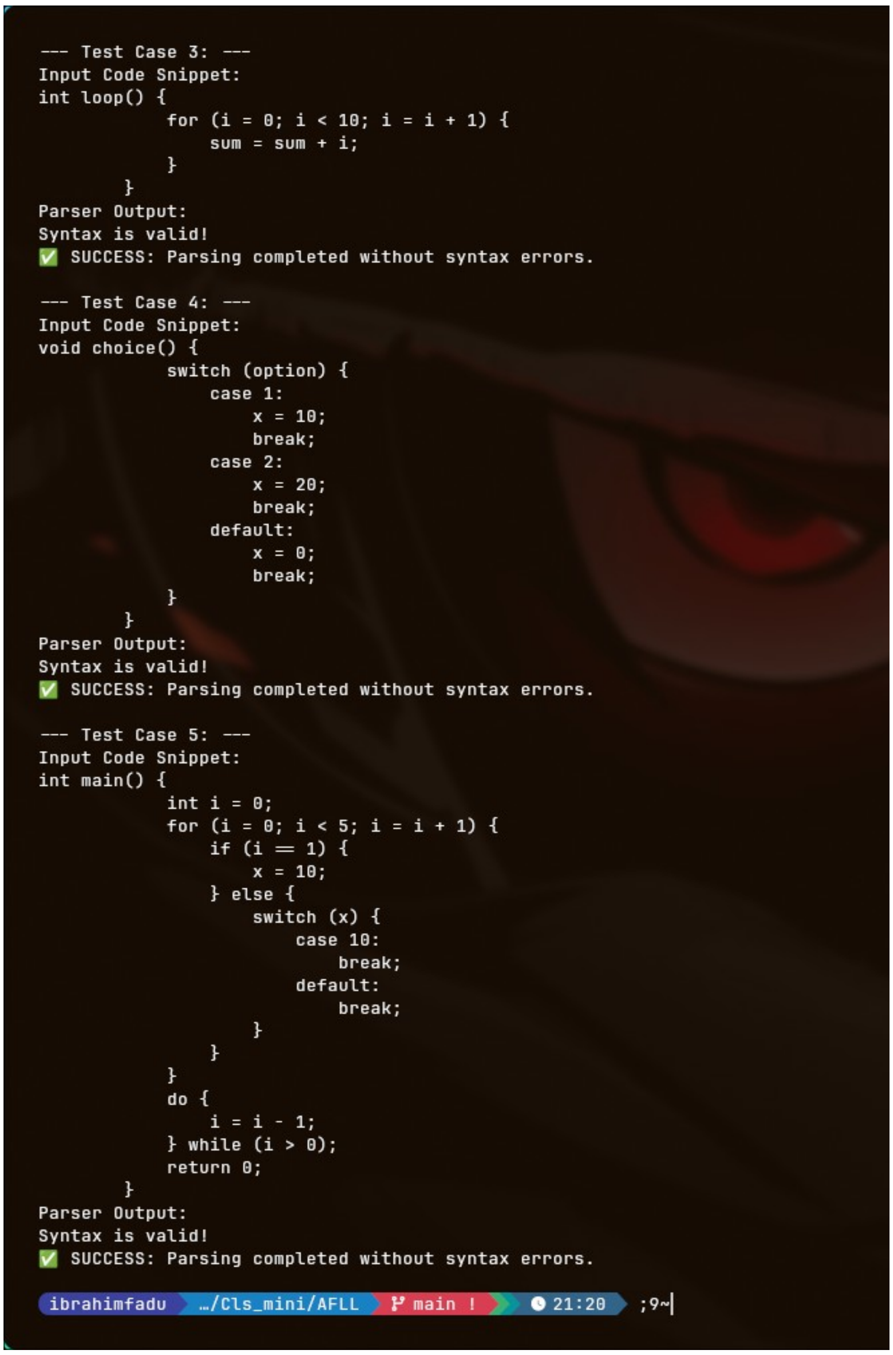
```
def p_error(p):
    else:
        print("Syntax error at EOF")

# Build the parser
parser = yacc.yacc()

# --- Test the parser with the chosen constructs ---
if __name__ == '__main__':
    print("=" * 50)
    print("TESTING C++ CONSTRUCTS WITH PLY")
    print("=" * 50)

    test_cases = [
        # Test 1: Do-While Loop
        """
        int main() {
            do {
                x = x - 1;
            } while (x > 0);
        }
        """,
        # Test 2: Nested If-Else
        """
        void check() {
            if (a == 10) {
                if (b < a) {
                    result = 1;
                } else {
                    result = 2;
                }
            } else {
                result = 3;
            }
        }
        """,
        # Test 3: For Loop
        """
        int loop() {
            for (i = 0; i < 10; i = i + 1) {
                sum = sum + i;
            }
        }
        """,
        # Test 4: Switch-Case
        """
        void choice() {
            switch (option) {
                case 1:
                    x = 10;
                    break;
                case 2:
                    x = 20;
                    break;
                default:
                    x = 0;
                    break;
```

maybe you need to run `:Codeium Auth`?

Screen Shot of Output



```
ibrahimfadu  …/Cls_mini/AFLL  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/AFLL  main !   21:20  ;9~
```