```
class Parser:
    def __init__(self, parsing_table, grammar):
        self.parsing_table = parsing_table
        self.grammar = grammar
        self.stack = ['0']
        self.input_string = ''

    def parse(self, input_string):
        self.stack = ['0']
        self.input_string = self.tokenize(input_string) + ['$']

        while True:
            state = self.stack[-1]
            current_input = self.input_string[0]
            action = self.parsing_table.get(state,
{}).get(current_input, '')

            print(f"Step: {state}, Stack: {self.stack}, Input:
{self.input_string}, Action: {action}")

            if action.startswith('S'):
                self.shift(action[1:])
            elif action.startswith('R'):
                if not self.reduce(action[1:]):
                    print("Error in reduction. Parsing halted.")
                    return False
            elif action == 'acc':
                print("String is accepted.")
                return True
            else:
                print("Error: Invalid action. String is not
accepted.")
                return False

    def tokenize(self, input_string):
        for symbol in ['+', '*', '(', ')', '$']:
            input_string = input_string.replace(symbol, f' {symbol} ')
        tokens = input_string.split()
        return tokens

    def shift(self, to_state):
        self.stack.append(self.input_string.pop(0))
        self.stack.append(to_state)


    def reduce(self, rule_number):
        rule = self.grammar[int(rule_number) - 1]
        rhs_symbols = rule[1].split()

        # Adjust the number of pops based on the actual symbols in RHS
        num_pops = len(rhs_symbols) * 2  # Each symbol has an
associated state

        # Ensure there are enough elements to pop
        if len(self.stack) < num_pops:
            print("Reduction error: Not enough elements in stack to
reduce.")
            return False

        for _ in range(num_pops):
            self.stack.pop()  # Pop both symbol and state

        top_state = self.stack[-1]
        self.stack.append(rule[0])  # Push LHS of the rule

        # Find the next state from the parsing table based on the LHS
of the rule
        goto_state = self.parsing_table.get(top_state,
{}).get(rule[0], '')
        if goto_state:
            self.stack.append(goto_state)
            return True
        else:
            print(f"Reduction error: No goto state found for {rule[0]}
in state {top_state}.")
            return False



# CFG rules in the form of (LHS, RHS)
grammar = [
    ('E', 'E + T'),
    ('E', 'T'),
    ('T', 'T * F'),
    ('T', 'F'),
    ('F', '( E )'),
    ('F', 'id')
]

# Parsing table and initialization remains the same
parsing_table = {
    '0': {'id': 'S5', '(': 'S4', 'E': '1', 'T': '2', 'F': '3'},
    '1': {'+': 'S6', '$': 'acc'},
    '2': {'+': 'R2', '*': 'S7', ')': 'R2', '$': 'R2'},
    '3': {'+': 'R4', '*': 'R4', ')': 'R4', '$': 'R4'},
    '4': {'id': 'S5', '(': 'S4', 'E': '8', 'T': '2', 'F': '3'},
    '5': {'+': 'R6', '*': 'R6', ')': 'R6', '$': 'R6'},
    '6': {'id': 'S5', '(': 'S4', 'T': '9', 'F': '3'},
    '7': {'id': 'S5', '(': 'S4', 'F': '10'},
    '8': {'+': 'S6', ')': 'S11'},
    '9': {'+': 'R1', '*': 'S7', ')': 'R1', '$': 'R1'},
    '10': {'+': 'R3', '*': 'R3', ')': 'R3', '$': 'R3'},
    '11': {'+': 'R5', '*': 'R5', ')': 'R5', '$': 'R5'},
}
# Initialize the parser with the parsing table and grammar
parser = Parser(parsing_table, grammar)

# Test strings as per project requirement
test_strings = ["( id + id ) * id $", "id * id $", "( id * ) $"]

# Parse each test string
for string in test_strings:
    print(f"\nParsing string: {string}")
    parser.parse(string)
```