PROJECT 2 Report File

CPSC 323-11: Group 4

Group Members

- Anar Otgonbaatar
- Andres Ugalde
- Daniel Felix
- Gary Samue

Explanation

- **Grammar**: The parser uses a CFG that supports variables **`(a)`**, operators **`(+, -, *, /)`**, and parentheses.
- **Parsing Table**: A nested dictionary used to store rules of the LL1 parsing table, where keys are non-terminals and lookahead terminals.
- **Stack-Based Parsing**:
    - Stack initialized with **`['$', 'E']`** (start symbol + end marker)
    - Input is appended with $ to denote the end
    - At each step, the parser compares the top of the stack with the current input symbol:
        - If they match, the symbol is popped.
        - If the top is non-terminal, the parser consults the table to expand using a production rule.
        - If no rule is found, it throws a syntax error.
- **Match Trace**: After every match or output step, the stack and remaining input are printed.

In-Built Functions Used

- **`print()`**: For displaying step by step parsing
- **`str()`**, **`list.reverse()`**, and **`''.join()`**: For stack formatting and production reversal
- No external libraries

Output Screenshots

```
Input: (a+a)*a
Stack                                          Input          Action
-------------------------------------------------
['$', 'E']                                     (a+a)*a$       Output E -> TQ
['$', 'Q', 'T']                                (a+a)*a$       Output T -> FR
['$', 'Q', 'R', 'F']                           (a+a)*a$       Output F -> (E)
['$', 'Q', 'R', ')', 'E', '(']                 (a+a)*a$       Match (
['$', 'Q', 'R', ')', 'E']                      a+a)*a$        Output E -> TQ
['$', 'Q', 'R', ')', 'Q', 'T']                 a+a)*a$        Output T -> FR
['$', 'Q', 'R', ')', 'Q', 'R', 'F']            a+a)*a$        Output F -> a
['$', 'Q', 'R', ')', 'Q', 'R', 'a']            a+a)*a$        Match a
['$', 'Q', 'R', ')', 'Q', 'R']                 +a)*a$         Output R -> ε
['$', 'Q', 'R', ')', 'Q']                      +a)*a$         Output Q -> +TQ
['$', 'Q', 'R', ')', 'Q', 'T', '+']            +a)*a$         Match +
['$', 'Q', 'R', ')', 'Q', 'T']                 a)*a$          Output T -> FR
['$', 'Q', 'R', ')', 'Q', 'R', 'F']            a)*a$          Output F -> a
['$', 'Q', 'R', ')', 'Q', 'R', 'a']            a)*a$          Match a
['$', 'Q', 'R', ')', 'Q', 'R']                 )*a$           Output R -> ε
['$', 'Q', 'R', ')', 'Q']                      )*a$           Output Q -> ε
['$', 'Q', 'R', ')']                           )*a$           Match )
['$', 'Q', 'R']                                *a$            Output R -> *FR
['$', 'Q', 'R', 'F', '*']                      *a$            Match *
['$', 'Q', 'R', 'F']                           a$             Output F -> a
['$', 'Q', 'R', 'a']                           a$             Match a
['$', 'Q', 'R']                                $              Output R -> ε
['$', 'Q']                                     $              Output Q -> ε
['$']                                          $              Accept
String Accepted ✅
```

```
Input: a*(a/a)
Stack                                          Input          Action
-----------------------------------------------
['$', 'E']                                     a*(a/a)$       Output E -> TQ
['$', 'Q', 'T']                                a*(a/a)$       Output T -> FR
['$', 'Q', 'R', 'F']                           a*(a/a)$       Output F -> a
['$', 'Q', 'R', 'a']                           a*(a/a)$       Match a
['$', 'Q', 'R']                                *(a/a)$        Output R -> *FR
['$', 'Q', 'R', 'F', '*']                      *(a/a)$        Match *
['$', 'Q', 'R', 'F']                           (a/a)$         Output F -> (E)
['$', 'Q', 'R', ')', 'E', '(']                 (a/a)$         Match (
['$', 'Q', 'R', ')', 'E']                      a/a)$          Output E -> TQ
['$', 'Q', 'R', ')', 'Q', 'T']                 a/a)$          Output T -> FR
['$', 'Q', 'R', ')', 'Q', 'R', 'F']            a/a)$          Output F -> a
['$', 'Q', 'R', ')', 'Q', 'R', 'a']            a/a)$          Match a
['$', 'Q', 'R', ')', 'Q', 'R']                 /a)$           Output R -> /FR
['$', 'Q', 'R', ')', 'Q', 'R', 'F', '/']       /a)$           Match /
['$', 'Q', 'R', ')', 'Q', 'R', 'F']            a)$            Output F -> a
['$', 'Q', 'R', ')', 'Q', 'R', 'a']            a)$            Match a
['$', 'Q', 'R', ')', 'Q', 'R']                 )$             Output R -> ε
['$', 'Q', 'R', ')', 'Q']                      )$             Output Q -> ε
['$', 'Q', 'R', ')']                           )$             Match )
['$', 'Q', 'R']                                $              Output R -> ε
['$', 'Q']                                     $              Output Q -> ε
['$']                                          $              Accept
String Accepted ✅
```

```
Input: a(a+a)
Stack                                          Input          Action
-----------------------------------------------
['$', 'E']                                     a(a+a)$        Output E -> TQ
['$', 'Q', 'T']                                a(a+a)$        Output T -> FR
['$', 'Q', 'R', 'F']                           a(a+a)$        Output F -> a
['$', 'Q', 'R', 'a']                           a(a+a)$        Match a
['$', 'Q', 'R']                                (a+a)$         Error: no rule for (R,
()
Syntax Error ❌
```