```
#CMPS 455 Assignment No. 7 Pt. 1
#Authors: Koppany Horvath
#Language: Python 3.6
#Task: Given the following CFG and the Predictive Parsing table. Write a program to trace input strings (1) (a+a)*a$ (2) a*(a/a)$ (3) a(a+a)$ Show the content of the stack after each match.

def matches(string):
      parseTable = {
            "E":{"a":"TQ", "+":None,  "-":None,  "*":None,  "/":None, "(":"TQ",  ")":None, "$":None},
            "Q":{"a":None, "+":"+TQ", "-":"-TQ", "*":None,  "/":None, "(":None,  ")":"",   "$":""},
            "T":{"a":"FR", "+":None,  "-":None,  "*":None,  "/":None, "(":"FR",  ")":None, "$":None},
            "R":{"a":None, "+":"",    "-":"",    "*":"*FR", "/":"/FR", "(":None,  ")":"",   "$":""},
            "F":{"a":"a",  "+":None,  "-":None,  "*":None,  "/":None, "(":"(E)", ")":None, "$":None}
      }
      stack = []
      curTerm = None
      curNonTerm = None
      done = False
      isGood = True

      stack.append("$")
      stack.append("E")

      while not done:
            curTerm = string[0] #read
            string = string[1:]

            while 1:
                  curNonTerm = stack.pop()
                  if curNonTerm in "a+-*/()$": #if it's a term, match
                        print("Match:", curNonTerm, " - ", "Stack:", stack)
                        if curNonTerm == "$": done = True #if it's the end then exit
                        break

                  p = parseTable[curNonTerm][curTerm]
                  if p == "": continue #if it's lambda, pop again
                  elif p == None: #if it's none, break with error
                        done = True
                        isGood = False
                        break

                  for x in p[::-1]: stack.append(x) #push in reverse order
      return isGood

for s in ["(a+a)*a$","a*(a/a)$","a(a+a)$"]:
      print("Working on string:", s)
      isMatch = matches(s)
      if isMatch: print("String matches grammar!")
```

```
48          else: print("Error: string does not match grammar!")
49          print()
50
51      """ Output:
52      Working on string: (a+a)*a$
53      Match: (  -  Stack: ['$', 'Q', 'R', ')', 'E']
54      Match: a  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'R']
55      Match: +  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'T']
56      Match: a  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'R']
57      Match: )  -  Stack: ['$', 'Q', 'R']
58      Match: *  -  Stack: ['$', 'Q', 'R', 'F']
59      Match: a  -  Stack: ['$', 'Q', 'R']
60      Match: $  -  Stack: []
61      String matches grammar!
62
63      Working on string: a*(a/a)$
64      Match: a  -  Stack: ['$', 'Q', 'R']
65      Match: *  -  Stack: ['$', 'Q', 'R', 'F']
66      Match: (  -  Stack: ['$', 'Q', 'R', ')', 'E']
67      Match: a  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'R']
68      Match: /  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'R', 'F']
69      Match: a  -  Stack: ['$', 'Q', 'R', ')', 'Q', 'R']
70      Match: )  -  Stack: ['$', 'Q', 'R']
71      Match: $  -  Stack: []
72      String matches grammar!
73
74      Working on string: a(a+a)$
75      Match: a  -  Stack: ['$', 'Q', 'R']
76      Error: string does not match grammar!
77      """
```