**COMPILER MINI PROJECT**

**Predictive Parsing Analysis**

**CODE:**

from colorama import Fore, init

class PredictiveParser:

def \_\_init\_\_(self):

# self.non\_terminals = list(input("Enter the list of non-terminals >"))

# self.terminals = list(input("Enter the list of terminals >"))

# print("Use `@` for denoting upsilon.")

# rule\_count = int(input("Enter the number of rules you want to add > "))

# self.production\_rules = list()

# for i in range(rule\_count):

# self.production\_rules.append(input(f"Enter rule {i + 1} > ").replace(" ", ""))

# self.first = self.follow = dict()

# for non\_terminal in self.non\_terminals:

# self.first[non\_terminal] = list(input(f"Enter first({non\_terminal}) > "))

# for non\_terminal in self.non\_terminals:

# self.follow[non\_terminal] = list(input(f"Enter follow({non\_terminal}) > "))

self.non\_terminals = list("EGTUF")

self.terminals = list("+\*()a")

self.production\_rules = ["E->TG", "G->+TG", "G->@", "T->FU", "U->\*FU", "U->@", "F->(E)", "F->a"]

self.first = {"E":["(", "a"], "G":["+", "@"], "T":["(", "a"], "U":["\*", "@"], "F":["(", "a"]}

self.follow = {"E":[")", "$"], "G":[")", "$"], "T":[")", "$", "+"], "U":[")", "$", "+"], "F":[")", "$", "+", "\*"]}

def generate\_parsing\_table(self) -> dict[str, list[str]]:

parsing\_table = dict()

for non\_terminal in self.non\_terminals:

parsing\_table[non\_terminal] = [None for i in range(len(self.terminals) + 1)]

for production\_rule in self.production\_rules:

non\_terminal\_at\_left, remainder = production\_rule.split("->") if "->" in production\_rule else production\_rule.split("-")

if not (remainder[0].isupper() or remainder[0] == "@"):

parsing\_table[non\_terminal\_at\_left][self.terminals.index(remainder[0])] = production\_rule

else:

update\_locations = self.first[non\_terminal\_at\_left]

if "@" in update\_locations:

update\_locations.remove("@")

update\_locations += self.follow[non\_terminal\_at\_left]

for update\_location in update\_locations:

try:

position = self.terminals.index(update\_location)

except ValueError:

position = len(self.terminals)

if parsing\_table[non\_terminal\_at\_left][position] is not None:

continue

parsing\_table[non\_terminal\_at\_left][position] = production\_rule

return parsing\_table

def print\_parsing\_table(self, parsing\_table : dict[str, list[str]]):

init()

yellow = Fore.YELLOW

red = Fore.RED

green = Fore.GREEN

magenta = Fore.MAGENTA

print(f"{yellow}Non Terminal", end = "\t")

for terminal in self.terminals:

print(f"{yellow}{terminal}", end = "\t")

print(f"{yellow}$", end = "\n")

for entry in parsing\_table:

print(f"{yellow}{entry}", end = "\t\t")

for cell in parsing\_table[entry]:

color = green if cell is not None else magenta

print(f"{color}{cell}", end = "\t")

print(end = "\n")

print("\n\n\n")

if \_\_name\_\_ == '\_\_main\_\_':

predictive\_parser = PredictiveParser()

parsing\_table = predictive\_parser.generate\_parsing\_table()

predictive\_parser.print\_parsing\_table(parsing\_table)

**OUTPUT:**

