Design and Implementaion of Modern Compiler

Mini Project

Aim: Write a code to generate a predictive parsing table for a given set of production rules.

Description:

- **Predictive Parsing:**
  Predictive parsing uses a stack and a parsing table to parse the input and generate a parse tree. Both the stack and the input contains an end symbol $to denote that the stack is empty and the input is consumed. The parser refers to the parsing table to take any decision on the input and stack element combination

- **Python:**
  Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small- and large-scale projects .

Source code:

```
class PredictiveParser:

    def __init__(self)

        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"]}
```

```python
        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:
```

```python
                        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]]):
        print("Non Terminal", end = "\t")
        for terminal in self.terminals:
            print(terminal, end = "\t")
        print("$", end = "\n")


        for entry in parsing_table:
            print(entry, end = "\t\t")
            for cell in parsing_table[entry]:
                print(cell, end = "\t")
            print(end = "\n")


if __name__ == '__main__':
    predictive_parser = PredictiveParser()

    parsing_table = predictive_parser.generate_parsing_table()

    predictive_parser.print_parsing_table(parsing_table)
```

<u>Output:</u>

```
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\smsha\OneDrive\Desktop\SEMESTER-2\MINI PROJECT\Design and Implementaion of Modern Compiler\COMPILER MP.py
Non Terminal     +         *         (         )         a         $
E                None      None      E->TG     None      E->TG     None
G                G->+TG    None      None      G->@      None      G->@
T                None      None      T->FU     None      T->FU     None
U                U->@      U->*FU    None      U->@      None      U->@
F                None      None      F->(E)    None      F->a      None
>>> |
```