Design And Implementaionof Modern Compilers

MiniProject

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

Description: Predictive parsing:

1. A predictive parser is a recursive descent parser with no backtracking or backup.
2. It is a top-down parser that does not require backtracking.
3. At each step, the choice of the rule to be expanded is made upon the next terminal symbol.

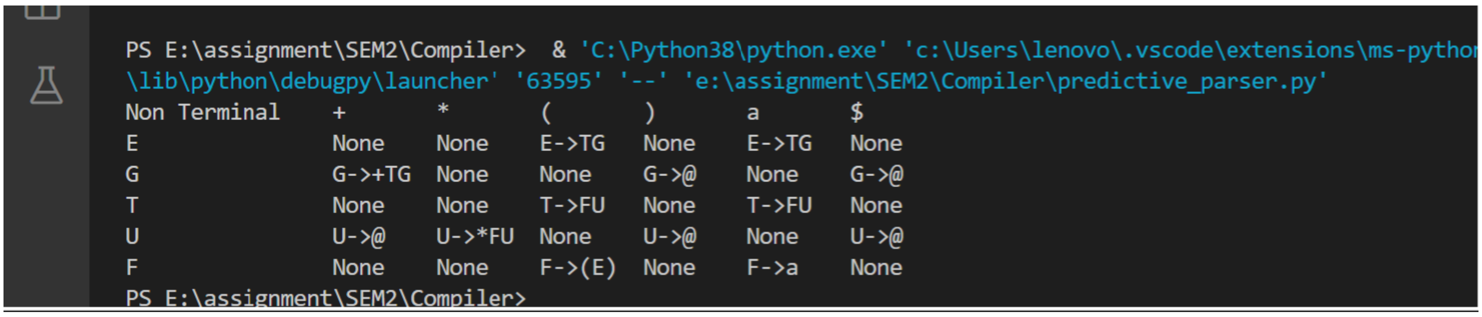
Source Code:-

```
from colorama import Fore, init


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"]}

                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_rul
e                               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]]'):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)
```

Output:-



```
PS E:\assignment\SEM2\Compiler>  & 'C:\Python38\python.exe' 'c:\Users\lenovo\.vscode\extensions\ms-pytho
\lib\python\debugpy\launcher' '63595' '--' 'e:\assignment\SEM2\Compiler\predictive_parser.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
PS E:\assignment\SEM2\Compiler>
```