Design and Implementation of Modern Compilers

Mini Project

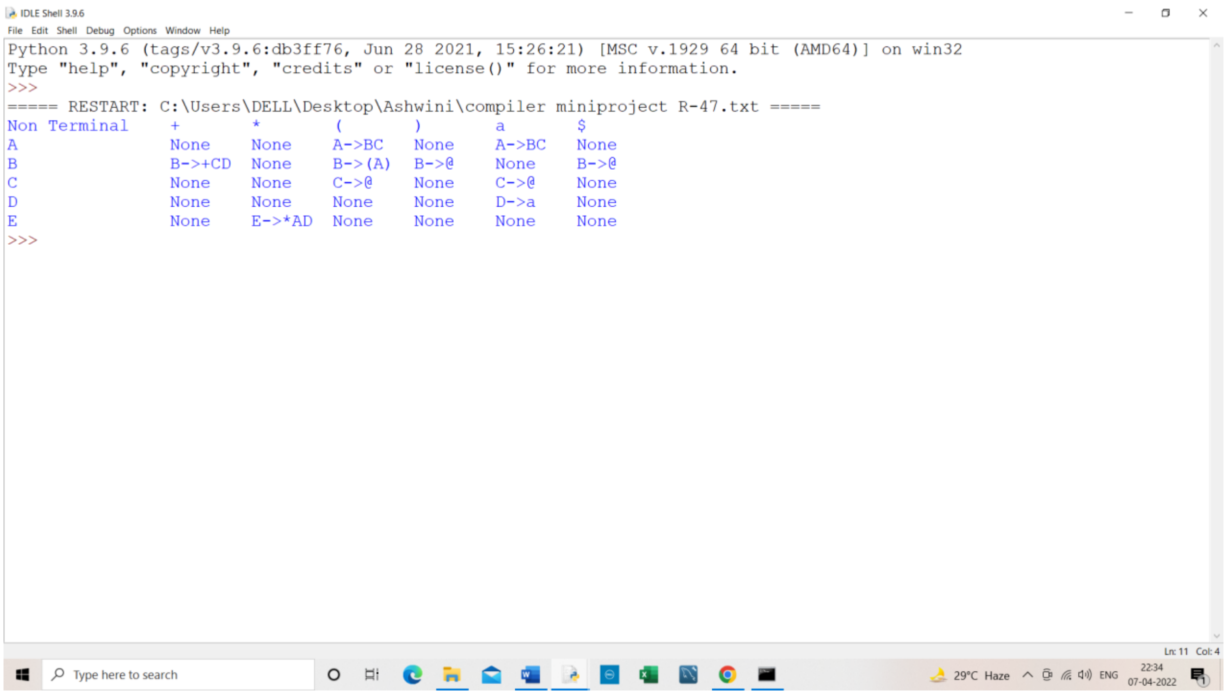
Aim: Design Predictive parser for given language.

Predictive parsing: It is a special form of recursive descent parsing, where no backtracking is required. It is a top-down parser.

Code:

```
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("ABCDE")
self.terminals = list("+*()a")
self.production_rules = ["A->BC", "B->+CD", "C->@", "C->DE", "E->*AD",
"B->@", "B->(A)", "D->a"]
self.first = {"A":["(", "a"], "B":["+", "@"], "C":["(", "a"], "D":["*", "@"], "E":["(",
"a"]}
self.follow = {"A":[")", "$"], "B":[")", "$"], "C":[")", "$", "+"], "D":[")", "$", "+"],
"E":[")", "$", "+", "*"]}



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]]):
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:



```
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DELL\Desktop\Ashwini\compiler miniproject R-47.txt =====
Non Terminal    +       *       (       )       a       $
A               None    None    A->BC   None    A->BC   None
B               B->+CD  None    B->(A)  B->@    None    B->@
C               None    None    C->@    None    C->@    None
D               None    None    None    None    D->a    None
E               None    E->*AD  None    None    None    None
>>>
```