**Shri Ramdeobaba College of Engineering and Management, Nagpur**

**Department of Computer Science and Engineering**

**Session: 2023-2024**

**Compiler Design Lab**

**PRACTICAL No. 3**

**Name : Kapil Gattani**

**Roll No : 44**

**Topic:** Parser Construction

**Language to be used:** Python

**Aim:**

1. **Write a program to find FIRST for any grammar. All the following rules of FIRST must be implemented.**

For a generalized grammar: A 🡪 XY

FIRST (A) = FIRST (XY)

=  if  is the terminal symbol (Rule-1)

= FIRST () if  is a non-terminal and FIRST () does not contain  (Rule-2)

= FIRST () -  ∪ FIRST (XY) if a is a non-terminal and FIRST () contains  (Rule-3)

**Input:** Grammar rules from a file or from console entered by user.

**Following inputs can be used:**

Batch A1:

A🡪 SB | B

S🡪 a | Bc | 

B🡪 b | d

Batch A2:

S🡪 A | BC

A🡪 a | b

B🡪 p | 

C🡪 c

Batch A4:

S🡪 ABC | C

A🡪 a | bB | 

B🡪 p | 

C🡪 c

Batch A3:

S🡪 AB | C

A🡪 a | b | 

B🡪 p | 

C🡪 c

**Implementation:** FIRST rules

**Output:** FIRST information for each non-terminal

1. **Calculate Follow for the given grammar and Construct the LL (1) parsing table using the FIRST and FOLLOW.**

**Submission Format:** It should contain- Aim, hand solved numerical (batch specific), code, and execution screen shot.

**Code :**

from collections import defaultdict

from tabulate import tabulate

class LLParser:

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

        self.productions = productions

        self.first = defaultdict(set)

        self.follow = defaultdict(set)

        self.start\_symbol = list(self.productions.keys())[0]

        self.calculate\_first()

        self.calculate\_follow()

        self.construct\_parsing\_table()

    def calculate\_first(self):

        for nt, rhs in reversed(self.productions.items()):

            for alt in rhs:

                for s in alt:

                    if s.isupper():

                        if 'e' not in self.first[s]:

                            self.first[nt].update(self.first[s])

                            if 'e' in self.first[nt]:

                                self.first[nt].remove('e')

                            break

                        self.first[nt].update(self.first[s])

                    elif s != 'e':

                        self.first[nt].add(s)

                        if 'e' in self.first[nt]:

                            self.first[nt].remove('e')

                        break

                    else:

                        self.first[nt].add('e')

    def calculate\_follow(self):

        self.follow[self.start\_symbol].add('$')

        while True:

            old\_follow = dict(self.follow)

            for nt, rhs in self.productions.items():

                for production in rhs:

                    for index, letter in enumerate(production[::-1]):

                        if index == 0:

                            if letter.isupper():

                                self.follow[letter].update(self.follow[nt])

                        else:

                            if letter.isupper():

                                temp = index-1

                                while temp >= 0:

                                    if list(reversed(production))[temp].islower():

                                        self.follow[letter].add(

                                            list(reversed(production))[temp])

                                        break

                                    else:

                                        if 'e' not in self.first[list(reversed(production))[temp]]:

                                            self.follow[letter].update(

                                                self.first[list(reversed(production))[temp]])

                                            break

                                    temp -= 1

            if old\_follow == self.follow:

                break

    def construct\_parsing\_table(self):

        terminals = set()

        non\_terminals = set(self.productions.keys())

        for nt, rhs in self.productions.items():

            for alt in rhs:

                for symbol in alt:

                    if not symbol.isupper() and symbol != 'e':

                        terminals.add(symbol)

        terminals.add('$')

        self.parsing\_table = defaultdict(lambda: defaultdict(str))

        for nt in non\_terminals:

            for t in terminals:

                for prod in self.productions[nt]:

                    for letter in prod:

                        if prod == 'e' and (t in self.follow[nt] or t == '$'):

                            self.parsing\_table[nt][t] = f"{nt}->e"

                            break

                        elif prod[0] == t or (prod[0].isupper() and t in self.first[prod[0]]):

                            self.parsing\_table[nt][t] = f"{nt}->{prod}"

                            break

                        elif letter.isupper() and 'e' in self.first[letter] and len(prod) > 1 and (prod.index(letter) + 1) < len(prod) and t in self.first[prod[prod.index(letter) + 1]]:

                            self.parsing\_table[nt][t] = f"{nt}->{prod}"

                            if 'e' not in self.first[letter]:

                                break

    def print\_parsing\_table(self):

        headers = sorted(list(self.parsing\_table['S'].keys()))

        rows = sorted(list(self.productions.keys()))

        table = [[self.parsing\_table[nt][t] for t in headers] for nt in rows]

        print(tabulate(table, headers=headers, showindex=rows, tablefmt='grid'))

def main():

    a2 = {

        'S': ['AB', 'C'],

        'A': ['a', 'b','e'],

        'B': ['p', 'e'],

        'C': ['c']

    }

    parser = LLParser(a2)

    # Print first sets

    print("First sets:")

    for nt, first\_set in parser.first.items():

        print(f"First({nt}): {first\_set}")

    # Print follow sets

    print("\nFollow sets:")

    for nt, follow\_set in parser.follow.items():

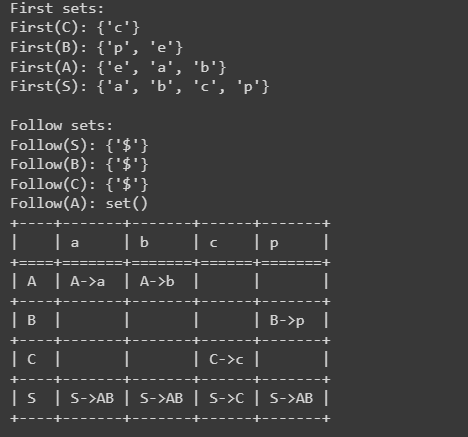
        print(f"Follow({nt}): {follow\_set}")

    parser.print\_parsing\_table()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output :**

****