Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering

Compiler Design Lab

PRACTICAL No. 3

Name: Gaurav Kedia **Rollno: 39** **Branch: AIML** **Batch: E2**

Topic: Parser Construction

Platform: Windows or Linux

Language to be used: Python or Java (based on the companies targeted for placement)

Aim:

(A) Write a program to find FIRST for any grammar. All the following rules of FIRST must be implemented.

For a generalized grammar: A → αXY

FIRST (A) = FIRST (αXY)
= α if α is the terminal symbol (Rule-1)
= FIRST (α) if α is a non-terminal and FIRST (α) does not contain ε (Rule-2)
= FIRST (α) - ε ∪ FIRST (XY) if a is a non-terminal and FIRST (α) contains ε (Rule-3)

Input: Grammar rules from a file or from console entered by user.
Following inputs can be used:

Batch E1:
A→ SB | B
S→ a | Bc | ε
B→ b | d

Batch E2:
S→ A | BC
A→ a | b
B→ p | ε
C→ c

Batch E3:
S→ AB | C
A→ a | b | ε
B→ p | ε
C→ c

Batch E4:
S→ ABC | C
A→ a | bB | ε
B→ p | ε
C→ c

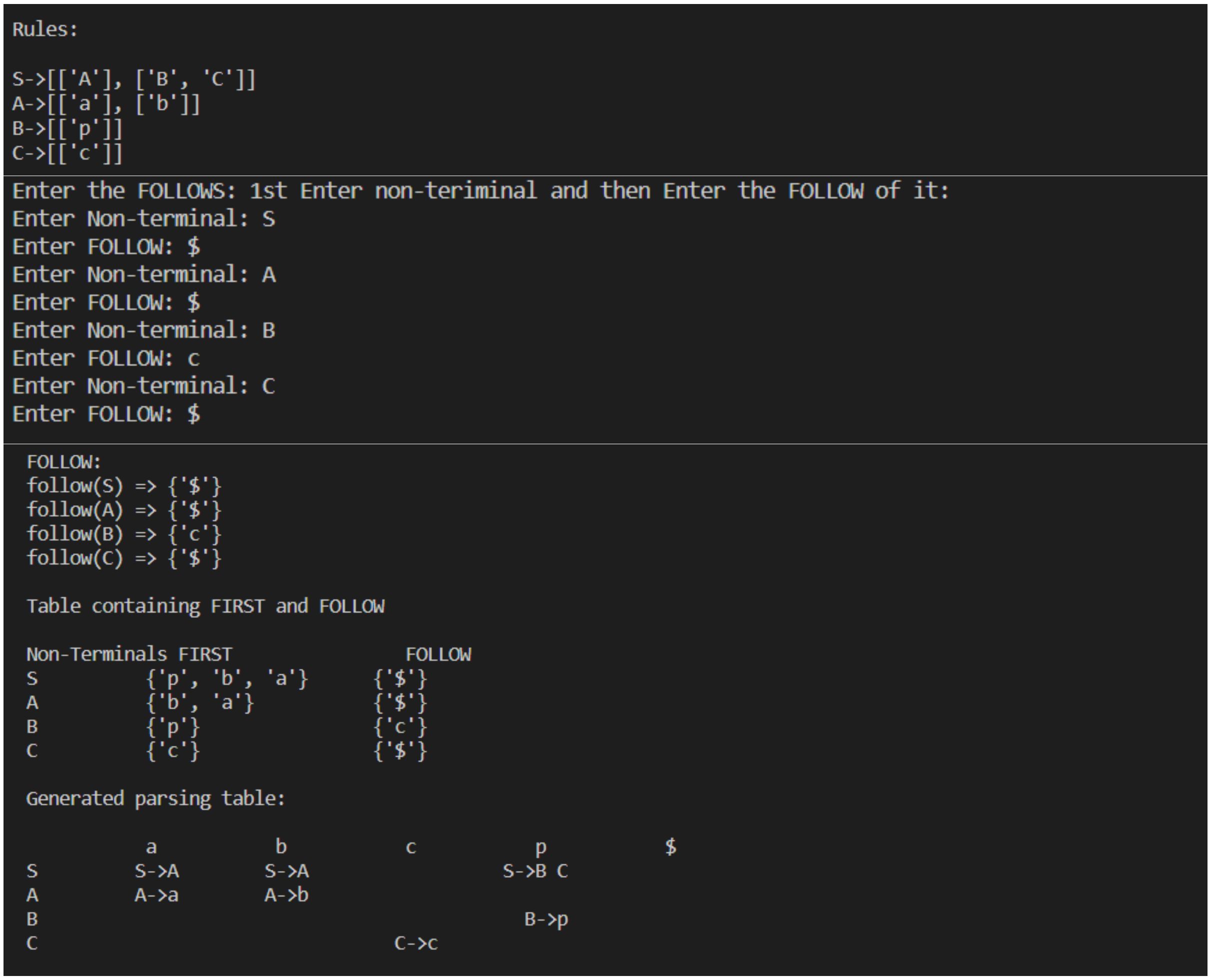
Implementation: FIRST rules
Output: FIRST information for each non-terminal

(B) Calculate Follow for the given grammar manually, input the follow information and Construct the LL (1) parsing table using the FIRST and FOLLOW values computed above.

Program:

```
def first(rule):
  global rules, nonterm_userdef, \
    term_userdef, diction, firsts
  if len(rule) != 0 and (rule is not None):
        if rule[0] in term_userdef:
            return rule[0]
        elif rule[0] == '#':
            return '#'

    if len(rule) != 0:
        if rule[0] in list(diction.keys()):
            fres = []
            rhs_rules = diction[rule[0]]
            for itr in rhs_rules:
                indivRes = first(itr)
                if type(indivRes) is list:
                    for i in indivRes:
                        fres.append(i)
                else:
                    fres.append(indivRes)

            if '#' not in fres:
                return fres
            else:
                newList = []
                fres.remove('#')
                if len(rule) > 1:
                    ansNew = first(rule[1:])
                    if ansNew != None:
                        if type(ansNew) is list:
                            newList = fres + ansNew
                        else:
                            newList = fres + [ansNew]
                    else:
                        newList = fres
                    return newList
                fres.append('#')
                return fres


def computeAllFirsts():
    global rules, nonterm_userdef, \
        term_userdef, diction, firsts
    for rule in rules:
        k = rule.split("->")
        k[0] = k[0].strip()
        k[1] = k[1].strip()
        rhs = k[1]
        multirhs = rhs.split('|')
        for i in range(len(multirhs)):
            multirhs[i] = multirhs[i].strip()
            multirhs[i] = multirhs[i].split()
        diction[k[0]] = multirhs

    print(f"\nRules: \n")
    for y in diction:
        print(f"{y}->{diction[y]}")
    print(f"\nAfter removing of left recursion:\n")
    diction = removeLeftRecursion(diction)
    for y in diction:
        print(f"{y}->{diction[y]}")
    print("\nAfter removing left factoring:\n")

    diction = LeftFactoring(diction)
    for y in diction:
        print(f"{y}->{diction[y]}")

    for y in list(diction.keys()):
        t = set()
        for sub in diction.get(y):
            res = first(sub)
            if res != None:
                if type(res) is list:
                    for u in res:
                        t.add(u)
                else:
                    t.add(res)

        firsts[y] = t

    print("\nFIRST: ")
    key_list = list(firsts.keys())
    index = 0
    for gg in firsts:
        print(f"first({key_list[index]}) "
              f"=> {firsts.get(gg)}")
        index += 1

def enterFollow():
    follow1={}
    global temp
    global temp1
    temp=""
    temp=""
    print("Enter the FOLLOWS: 1st Enter non-teriminal and then Enter the FOLLOW of it:")

    for i in range(0,4):
        temp = input("Enter Non-terminal: ")
        temp1 = input("Enter FOLLOW: ")


def createParseTable():
    import copy
    global diction, firsts, follows, term_userdef
    print("\nTable containing FIRST and FOLLOW\n")

    mx_len_first = 0
    mx_len_fol = 0
    for u in diction:
        k1 = len(str(firsts[u]))
        k2 = len(str(follows[u]))
        if k1 > mx_len_first:
            mx_len_first = k1
        if k2 > mx_len_fol:
            mx_len_fol = k2

    print(f"{{:<{10}}} "
          f"{{:<{mx_len_first + 5}}} "
          f"{{:<{mx_len_fol + 5}}}"
          .format("Non-Terminals", "FIRST", "FOLLOW"))
    for u in diction:
        print(f"{{:<{10}}} "
              f"{{:<{mx_len_first + 5}}} "
              f"{{:<{mx_len_fol + 5}}}"
              .format(u, str(firsts[u]), str(follows[u])))

    ntlist = list(diction.keys())
    terminals = copy.deepcopy(term_userdef)
    terminals.append('$')

    mat = []
    for x in diction:
        row = []
        for y in terminals:
            row.append('')
        mat.append(row)

    grammar_is_LL = True

    for lhs in diction:
        rhs = diction[lhs]
        for y in rhs:
            res = first(y)
            if '#' in res:
                if type(res) == str:
                    firstFollow = []
                    fol_op = follows[lhs]
                    if fol_op is str:
                        firstFollow.append(fol_op)
                    else:
                        for u in fol_op:
                            firstFollow.append(u)
                    res = firstFollow
                else:
                    res.remove('#')
                    res = list(res) +\
                        list(follows[lhs])
            ttemp = []
            if type(res) is str:
                ttemp.append(res)
                res = copy.deepcopy(ttemp)
            for c in res:
                xnt = ntlist.index(lhs)
                yt = terminals.index(c)
                if mat[xnt][yt] == '':
                        mat[xnt][yt] = mat[xnt][yt] \
                            + f"{lhs}->{' '.join(y)}"
                    else:
                        if f"{lhs}->{y}" in mat[xnt][yt]:
                            continue
                        else:
                            grammar_is_LL = False
                            mat[xnt][yt] = mat[xnt][yt] \
                                + f",{lhs}->{' '.join(y)}"

        print("\nGenerated parsing table:\n")
        frmt = "{:>12}" * len(terminals)
        print(frmt.format(*terminals))

        j = 0
        for y in mat:
            frmt1 = "{:>12}" * len(y)
            print(f"{ntlist[j]} {frmt1.format(*y)}")
            j += 1

        return (mat, grammar_is_LL, terminals)

    sample_input_string = None

    rules = ["S -> A | B C",
             "A -> a | b",
             "B -> p",
             "C -> c"]

    nonterm_userdef = ['S', 'A', 'B', 'C']
    term_userdef = ['a', 'b', 'c', 'p']
    sample_input_string = "p c"

    diction = {}
    firsts = {}
    follows = {}

    computeAllFirsts()
    start_symbol = list(diction.keys())[0]
    computeAllFollows()

    (parsing_table, result, tabTerm) = createParseTable()

    if sample_input_string != None:
        validity = validateStringUsingStackBuffer(parsing_table, result,
                                                  tabTerm, sample_input_string,
                                                  term_userdef, start_symbol)
        print(validity)
    else:
        print("\nNo input String detected")
```

OUTPUT:



```
Rules:

S->[['A'], ['B', 'C']]
A->[['a'], ['b']]
B->[['p']]
C->[['c']]
```

```
Enter the FOLLOWS: 1st Enter non-teriminal and then Enter the FOLLOW of it:
Enter Non-terminal: S
Enter FOLLOW: $
Enter Non-terminal: A
Enter FOLLOW: $
Enter Non-terminal: B
Enter FOLLOW: c
Enter Non-terminal: C
Enter FOLLOW: $
```

```
FOLLOW:
follow(S) => {'$'}
follow(A) => {'$'}
follow(B) => {'c'}
follow(C) => {'$'}

Table containing FIRST and FOLLOW

Non-Terminals FIRST                FOLLOW
S          {'p', 'b', 'a'}      {'$'}
A          {'b', 'a'}           {'$'}
B          {'p'}                {'c'}
C          {'c'}                {'$'}

Generated parsing table:

           a           b           c           p           $
S         S->A        S->A                    S->B C
A         A->a        A->b
B                                               B->p
C                                 C->c
```