Name: Diksha Gupta
Roll No. 27

PRACTICAL NO. 3

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.

Batch A1:
A→ SB | B
S→ a | Bc | ε
B→ b | d

Batch A2:
S→ A | BC
A→ a | b
B→ p | ε
C→ c

Batch A3:
S→ AB | C
A→ a | b | ε
B→ p | ε
C→ c

Batch A4:
S→ ABC | C
A→ a | bB | ε
B→ p | ε
C→ c

Following inputs can be used:

Implementation: FIRST rules

Output: FIRST information for each non-terminal

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

CODE :

```
from collections import OrderedDict

def isterminal(char):
    if(char.isupper() or char == "`"):
        return False
    else:
        return True

def insert(grammar, lhs, rhs):
    if(lhs in grammar and rhs not in grammar[lhs] and grammar[lhs] != "nul
l"):
        grammar[lhs].append(rhs)
    elif(lhs not in grammar or grammar[lhs] == "null"):
        grammar[lhs] = [rhs]
    return grammar

def first(lhs, grammar, grammar_first):
    rhs = grammar[lhs]
    for i in rhs:
        k = 0
        flag = 0
        current = []
        confirm = 0
        flog = 0
        if(lhs in grammar and "`" in grammar_first[lhs]):
            flog = 1
        while(1):
            check = []
            if(k>=len(i)):
                if(len(current)==0 or flag == 1 or confirm == k or flog ==
 1):
                    grammar_first = insert(grammar_first, lhs, "`")
                break
            if(i[k].isupper()):
                if(grammar_first[i[k]] == "null"):
                    grammar_first = first(i[k], grammar, grammar_first)
                for j in grammar_first[i[k]]:
                    grammar_first = insert(grammar_first, lhs, j)
                    check.append(j)
            else:
                grammar_first = insert(grammar_first, lhs, i[k])
                check.append(i[k])
            if(i[k]=="`"):
                flag = 1
            current.extend(check)
            if("`" not in check):
                if(flog == 1):
                    grammar_first = insert(grammar_first, lhs, "`")
                break
```

```
            else:
                confirm += 1
                k+=1
                grammar_first[lhs].remove("`")
    return(grammar_first)

def rec_follow(k, next_i, grammar_follow, i, grammar, start, grammar_first
, lhs):
    if(len(k)==next_i):
        if(grammar_follow[i] == "null"):
            grammar_follow = follow(i, grammar, grammar_follow, start)
        for q in grammar_follow[i]:
            grammar_follow = insert(grammar_follow, lhs, q)
    else:
        if(k[next_i].isupper()):
            for q in grammar_first[k[next_i]]:
                if(q=="`"):
                    grammar_follow = rec_follow(k, next_i+1, grammar_follo
w, i, grammar, start, grammar_first, lhs)
                else:
                    grammar_follow = insert(grammar_follow, lhs, q)
        else:
            grammar_follow = insert(grammar_follow, lhs, k[next_i])

    return(grammar_follow)

def follow(lhs, grammar, grammar_follow, start):
    for i in grammar:
        j = grammar[i]
        for k in j:
            if(lhs in k):
                next_i = k.index(lhs)+1
                grammar_follow = rec_follow(k, next_i, grammar_follow, i,
grammar, start, grammar_first, lhs)
    if(lhs==start):
        grammar_follow = insert(grammar_follow, lhs, "$")
    return(grammar_follow)

def show_dict(dictionary):
    for key in dictionary.keys():
        print(key+"  :  ", end = "")
        for item in dictionary[key]:
            if(item == "`"):
                print("Epsilon, ", end = "")
            else:
                print(item+", ", end = "")
        print("\b\b")

def get_rule(non_terminal, terminal, grammar, grammar_first):
    for rhs in grammar[non_terminal]:
        #print(rhs)
```

```
        for rule in rhs:
            if(rule == terminal):
                string = non_terminal+"~"+rhs
                return string

            elif(rule.isupper() and terminal in grammar_first[rule]):
                string = non_terminal+"~"+rhs
                return string

def generate_parse_table(terminals, non_terminals, grammar, grammar_first,
 grammar_follow):
    parse_table = [[""]*len(terminals) for i in range(len(non_terminals))]

    for non_terminal in non_terminals:
        for terminal in terminals:
            if terminal in grammar_first[non_terminal]:
                rule = get_rule(non_terminal, terminal, grammar, grammar_f
irst)

            elif("`" in grammar_first[non_terminal] and terminal in gramma
r_follow[non_terminal]):
                rule = non_terminal+"~`"

            elif(terminal in grammar_follow[non_terminal]):
                rule = "Sync"

            else:
                rule = ""

            parse_table[non_terminals.index(non_terminal)][terminals.index
(terminal)] = rule

    return(parse_table)

def display_parse_table(parse_table, terminal, non_terminal):
    print("\t\t\t\t",end = "")
    for terminal in terminals:
        print(terminal+"\t\t", end = "")
    print("\n\n")

    for non_terminal in non_terminals:
        print("\t\t"+non_terminal+"\t\t", end = "")
        for terminal in terminals:
            print(parse_table[non_terminals.index(non_terminal)][terminals
.index(terminal)]+"\t\t", end = "")
        print("\n")

# Main_Driver

grammar = OrderedDict()
grammar_first = OrderedDict()
```

```
grammar_follow = OrderedDict()

f = open('grammar.txt')
for i in f:
    i = i.replace("\n", "")
    lhs = ""
    rhs = ""
    flag = 1
    for j in i:
        if(j=="~"):
            flag = (flag+1)%2
            continue
        if(flag==1):
            lhs += j
        else:
            rhs += j
    grammar = insert(grammar, lhs, rhs)
    grammar_first[lhs] = "null"
    grammar_follow[lhs] = "null"

print("Grammar\n")
show_dict(grammar)

for lhs in grammar:
    if(grammar_first[lhs] == "null"):
        grammar_first = first(lhs, grammar, grammar_first)

print("\n\n\n")
print("First\n")
show_dict(grammar_first)


start = list(grammar.keys())[0]
for lhs in grammar:
    if(grammar_follow[lhs] == "null"):
        grammar_follow = follow(lhs, grammar, grammar_follow, start)

print("\n\n\n")
print("Follow\n")
show_dict(grammar_follow)


non_terminals = list(grammar.keys())
terminals = []

for i in grammar:
    for rule in grammar[i]:
        for char in rule:

            if(isterminal(char) and char not in terminals):
                terminals.append(char)
```

```
terminals.append("$")

print("\n\n\n\n\t\t\t\t\t\t\tParse Table\n\n")
parse_table = generate_parse_table(terminals, non_terminals, grammar, gram
mar_first, grammar_follow)
display_parse_table(parse_table, terminals, non_terminals)
```

OUTPUT :

```
Grammar

S  :  A, BC
A  :  a, b
B  :  p, Epsilon
C  :  c


First

S  :  a, b, p, c
A  :  a, b
B  :  p, Epsilon
C  :  c


Follow

S  :  $
A  :  $
B  :  c
C  :  $

                                        Parse Table

                    a               b               p               c               $

          S         S~A             S~A             S~BC            S~BC            Sync

          A         A~a             A~b                                             Sync

          B                                         B~p             B~`

          C                                                         C~c             Sync
```