**Name: Bhavesh Kewalramani**

**Roll No.: A-25**

**Practical No. 3**

**Theory**

**LL(1) Parsing:**   
Here the 1st **L** represents that the scanning of the Input will be done from Left to Right manner and the second **L** shows that in this parsing technique we are going to use Left most Derivation Tree. And finally, the **1** represents the number of look-ahead, which means how many symbols are you going to see when you want to make a decision.

**Algorithm to construct LL(1) Parsing Table:**

**Step 1:**First check for left recursion in the grammar, if there is left recursion in the grammar remove that and go to step 2.

**Step 2:**Calculate First() and Follow() for all non-terminals.

1. **First():** If there is a variable, and from that variable, if we try to drive all the strings then the beginning Terminal Symbol is called the First.
2. Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

**Step 3:**For each production A –> α. (A tends to alpha)

1. Find First(α) and for each terminal in First(α), make entry A –> α in the table.
2. If First(α) contains ε (epsilon) as terminal than, find the Follow(A) and for each terminal in Follow(A), make entry A –> α in the table.
3. If the First(α) contains ε and Follow(A) contains $ as terminal, then make entry A –> α in the table for the $.  
   To construct the parsing table, we have two functions:

In the table, rows will contain the Non-Terminals and the column will contain the Terminal Symbols. All the **Null Productions** of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

**Practicals**

**Aim:**

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

**B**. Further, write a program to find Follow for the given grammar.

**C.** Construct the LL(1) parsing table using the FIRST and FOLLOW values computed

above.

**Program:**

from collections import OrderedDict

# ` symbol represents EPSILON

def EnterGrammar():

d = OrderedDict()

f = open("grammar.txt", "r")

print('Grammar\n'+f.read())

f.seek(0)

for line in f:

k = ""

for c in line:

if c != "~" and k == "":

d[c] = []

k = c

elif c != "~" and c != "\n":

d[k].append(c)

f.seek(0)

nonterminal = []

terminal = []

for line in f:

for c in line:

if c not in d.keys() and c!= "~" and c!= "\n" and c!= "`" and c!= "/" and c not in terminal:

terminal.append(c)

if c>='A' and c<='Z' and c not in nonterminal:

nonterminal.append(c)

return d,nonterminal,terminal

def Calculate\_First(rule, index):

first = ""

v = rule[index]

j = 1

for i in range(len(v)):

if v[i] == "/":

j = 1

elif j == 1:

if v[i] not in rule.keys():

if v[i] not in first and v[i] != "/":

first = first + v[i]

j = 0

else:

a = list(Calculate\_First(rule, v[i]))

while "`" in a and i+1 < len(v) and v[i+1] != "/":

a.remove("`")

if v[i+1] not in rule.keys():

a.append(v[i+1])

else:

a = list(set().union(a, Calculate\_First(rule, v[i+1])))

i += 1

a.extend(first)

first = "".join(list(set(a)))

j = 0

return first

def Calculate\_Follow(rule, n, start\_symbol):

follow = ""

if n == start\_symbol:

follow += "$"

for k, v in rule.items():

for i in range(len(v)):

if v[i] == n:

if i == len(v) - 1:

follow += Calculate\_Follow(rule, k, start\_symbol)

elif i + 1 < len(v) and v[i + 1] not in rule.keys() and v[i+1] != "/" and v[i+1] not in follow:

follow += v[i + 1]

elif i + 1 < len(v) and v[i+1] != "/" and v[i+1] not in follow:

a = []

for j in first[v[i + 1]]:

a.append(j)

if "`" in a:

a.remove("`")

a.append(Calculate\_Follow(d, v[i+1], start\_symbol))

follow += "".join(list(set("".join(a))))

elif k == start\_symbol:

follow += "$"

return follow

def parsingTable(rule,nonterminals,terminals,first,follow):

terminals.append('$')

#make table

parse\_table = [ ["\_"]\*(len(terminals) + 1) for i in range(len(nonterminals) + 1) ]

for i in range(len(parse\_table)):

for j in range(len(parse\_table[0])):

if i == 0 and j != 0:

parse\_table[i][j] = terminals[j-1]

if i != 0 and j == 0:

parse\_table[i][j] = nonterminals[i-1]

#fill table

for i in range(1,len(parse\_table)):

for j in range(1,len(parse\_table[0])):

if parse\_table[0][j] in first[parse\_table[i][0]]:

key = parse\_table[i][0]

for k,v in rule.items():

if k == key:

val = v

c = 0

flag = 0

while c < len(val) and flag == 0:

if val[c] in nonterminals:

if parse\_table[0][j] in first[val[c]]:

rhs = ''

for k in range(c, len(val)):

if val[k] == '/':

break

rhs += val[k]

ans = parse\_table[i][0] + '~' + rhs

parse\_table[i][j] = ans

flag = 1

elif val[c] == parse\_table[0][j]:

rhs = ''

for k in range(c,len(val)):

if val[k] == '/':

break

rhs += val[k]

ans = parse\_table[i][0] + '~' + rhs

parse\_table[i][j] = ans

flag = 1

else:

while val[c] != '/' and c < len(val):

c += 1

if c < len(val):

c += 1

elif parse\_table[0][j] in follow[parse\_table[i][0]] and '`' in first[parse\_table[i][0]]:

parse\_table[i][j] = parse\_table[i][0] + '~`'

elif parse\_table[0][j] in follow[parse\_table[i][0]]:

parse\_table[i][j] = '\_'

else:

pass

return parse\_table

rule,nonterminals,terminals = EnterGrammar()

start\_symbol = input("Enter the start symbol : ")

print()

print("Dictionary:",rule)

print()

first = OrderedDict()

for k, v in rule.items():

first[k] = []

first[k].extend(Calculate\_First(rule, k))

print("First:",first)

print()

follow = OrderedDict()

for k, v in rule.items():

follow[k] = []

follow[k].extend(Calculate\_Follow(rule, k, start\_symbol))

follow[k] = list(set(follow[k]))

print("Follow:", follow)

print()

parse\_table = parsingTable(rule,nonterminals,terminals,first,follow)

print("Parsing Table")

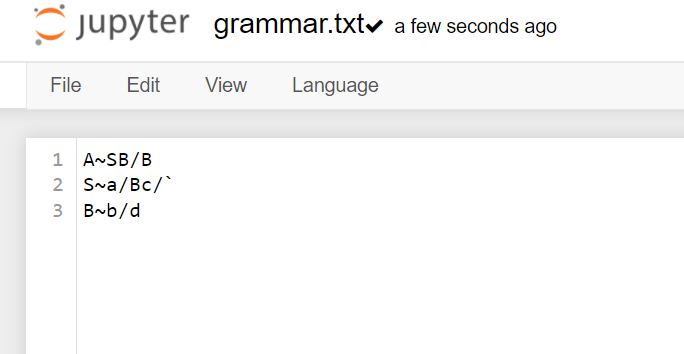
for i in range(len(parse\_table)):

for j in range(len(parse\_table[0])):

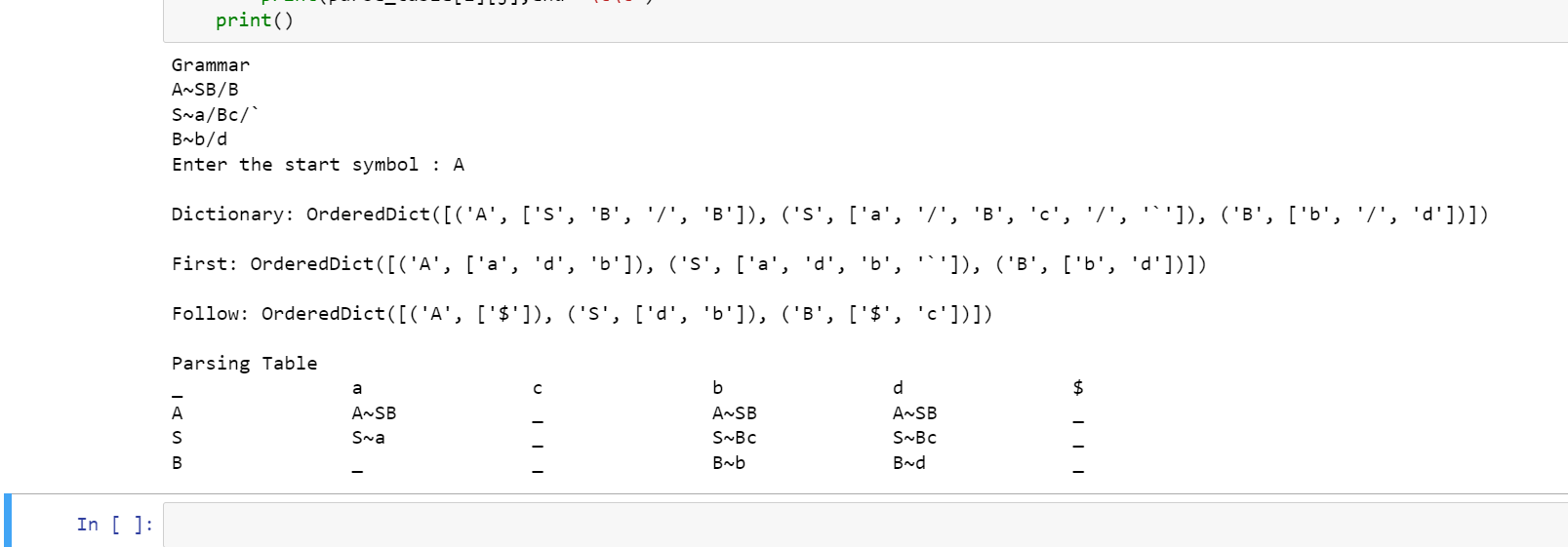
print(parse\_table[i][j],end="\t\t")

print()

**Input:**

****

**Output:**

****