**Name: Shreyas Namjoshi**

**Roll No: 53**

**Practical No. 3**

**Theory**

**FIRST Set :**

FIRST(X) for a grammar symbol X is the set of terminals that begin the strings derivable from X.

**Rules to compute FIRST set:**

1. If x is a terminal, then FIRST(x) = { ‘x’ }
2. If x-> Є, is a production rule, then add Є to FIRST(x).
3. If X->Y1 Y2 Y3….Yn is a production,
   1. FIRST(X) = FIRST(Y1)
   2. If FIRST(Y1) contains Є then FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }
   3. If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X).

**FOLLOW Set :**

**Follow(X)** to be the set of terminals that can appear immediately to the right of Non-Terminal X in some sentential form.

**Rules to compute FOLLOW set:**

1. FOLLOW(S) = { $ } // where S is the starting Non-Terminal

2. If A -> pBq is a production, where p, B and q are any grammar symbols,

then everything in FIRST(q) except Є is in FOLLOW(B).

3. If A->pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).

4. If A->pBq is a production and FIRST(q) contains Є,

then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)

**LL(1) Parsing:**

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](https://www.geeksforgeeks.org/removing-direct-and-indirect-left-recursion-in-a-grammar/) 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](https://www.geeksforgeeks.org/first-set-in-syntax-analysis/)(): 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](https://www.geeksforgeeks.org/follow-set-in-syntax-analysis/)(): 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:

**AIM :**

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

must be implemented.

Following inputs can be used:

Implementation: FIRST rules

Output: FIRST information for each non-terminal

(B) Construct the LL(1) parsing table using the FIRST values computed above and

consider Follow information as input from the user.

Batch B3: **# = Epsilon**

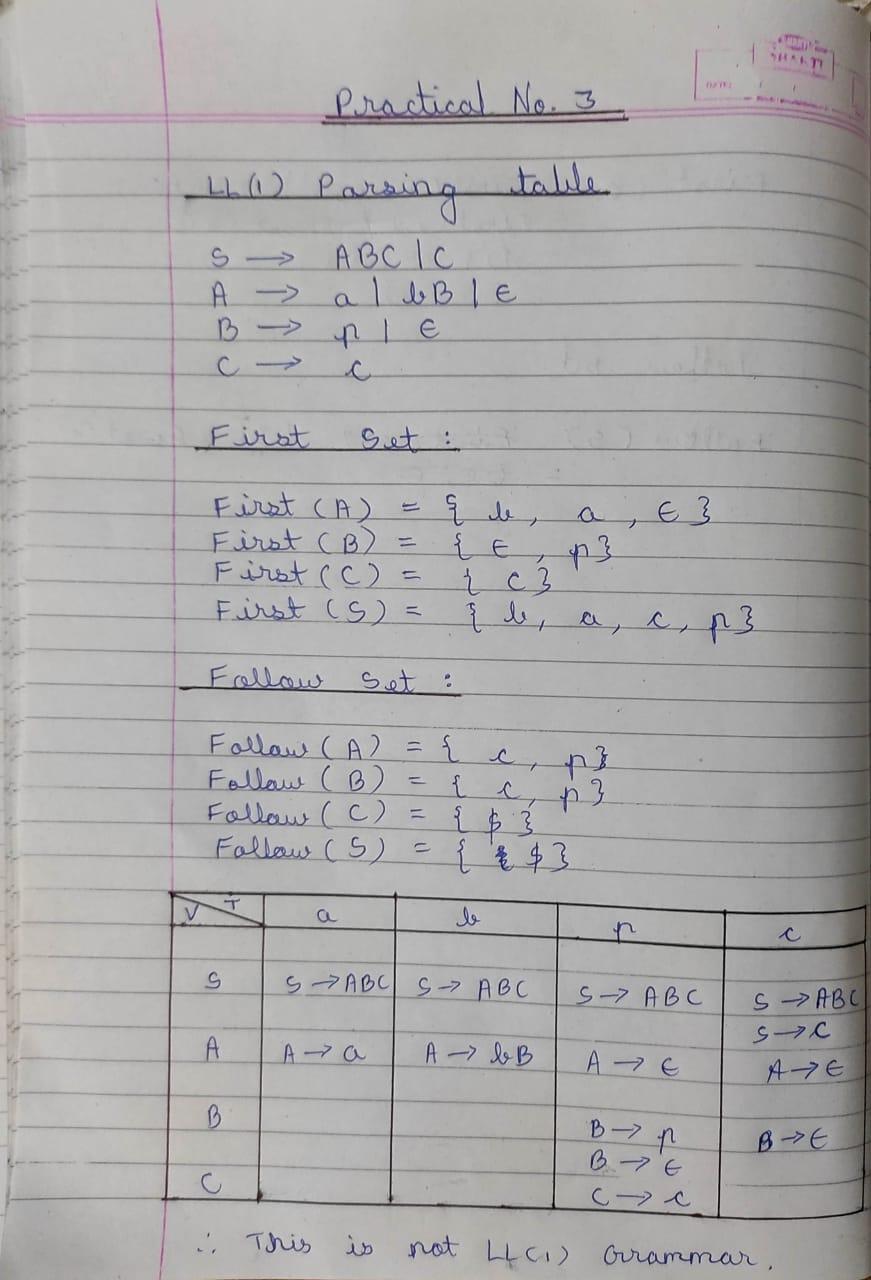
S -> AB | C

A -> a | b | #

B -> p | #

C -> c

**HAND WRITTEN EXAMPLE :**

****

**CODE :**

firstD = {}  
followD = {}  
eps = **"Є"  
  
def** CaluclateFirst(start):  
 first = set()  
  
 **for** v **in** rules[start]:  
 **for** each **in** v:  
 **if** eps **in** first:  
 first.remove(eps)  
  
 **if** each **in** ter:  
 first.add(each)  
  
 **elif** each == eps:  
 first.add(eps)  
  
 **elif** each **in** non\_ter:  
 first = first.union(CaluclateFirst(each))  
  
 **if** eps **not in** first:  
 **break** firstD[start] = first  
 **return** first  
  
  
**def** CalculateFollow(start):  
 follow = set()  
 **if** start == start\_symbol:  
 follow.add(**"$"**)  
 **for** k, r **in** rules.items():  
 *#print(start,'\t', r)* **for** each **in** r:  
 **if** start **in** each:  
 index = each.index(start)  
 **if** index != len(each)-1:  
 **for** i **in** range(index+1, len(each)):  
 **if** eps **in** follow:  
 follow.remove(eps)  
 followD[start] = follow  
 **if** each[i] **in** ter:  
 follow = follow.union(each[i])  
 followD[start] = follow  
 **else**:  
 follow = follow.union(firstD[each[i]])  
 followD[start] = follow  
 **if** eps **not in** follow:  
 **break  
 else**:  
 **if** k **not in** followD:  
 follow = follow.union(CalculateFollow(k))  
 followD[start] = follow  
 **else**:  
 follow = follow.union(followD[k])  
 followD[start] = follow  
 **if** eps **in** follow **or** len(follow) == 0:  
 **if** eps **in** follow:  
 follow.remove(eps)  
 follow = follow.union(CalculateFollow(k))  
 followD[start] = follow  
 followD[start] = follow  
 **return** follow  
  
  
*#input terminal symbols*print(**"=========================================================================="**)  
print(**"Enter terminals:"**)  
ter = list(map(str, input().split()))  
print(**"=========================================================================="**)  
*#input non terminal symbols*print(**"Enter non terminals:"**)  
non\_ter = list(map(str, input().split()))  
print(**"=========================================================================="**)  
*#input start symbol*start\_symbol = input(**"Enter the starting symbol: "**)  
print(**"=========================================================================="**)  
*#input all production rules*no\_of\_productions = int(input(**"Enter no of productions: "**))  
productions = []  
print(**"=========================================================================="**)  
print(**"Enter the production rules:"**)  
**for** \_ **in** range(no\_of\_productions):  
 productions.append(input().replace(**"#"**, eps))  
  
rules = {}  
**for** p **in** productions:  
 r = p.split(**"->"**)  
 rules[r[0]] = r[1].split(**'|'**)  
print(rules)  
  
  
CaluclateFirst(start\_symbol)  
  
  
**for** start **in** non\_ter:  
 CaluclateFirst(start)  
print(**"=========================================================================="**)  
print(**"\tFIRST SET COMPUTATION TABLE\n"**)  
print(**"TERMINAL\t\t FIRST"**)  
**for** F **in** sorted(firstD):  
 print(**" "**,F,**"\t:\t"**,firstD[F])  
  
  
**for** start **in** non\_ter:  
 CalculateFollow(start)  
print(**"=========================================================================="**)  
print(**"\n\tFOLLOW SET COMPUTATION TABLE\n"**)  
print(**"TERMINAL\t\t FOLLOW"**)  
**for** F **in** sorted(followD):  
 print(**" "**,F,**"\t:\t"**,followD[F])  
  
  
**def** parsingTable(rules):  
 **for** symbol, prod **in** rules.items():  
 **for** each **in** prod:  
 t = set()  
 **for** e **in** each:  
 **if** e **in** non\_ter:  
 **if** eps **in** t:  
 t.remove(eps)  
 t = t.union(firstD[e])  
 **if** eps **not in** t:  
 **break  
 else**:  
 t = t.union(e)  
 **break  
 if** eps **in** t:  
 t.remove(eps)  
 t = t.union(followD[symbol])  
 table[symbol].append([{symbol+**'->'**+each: t}])  
  
table = dict()  
**for** each **in** non\_ter:  
 table[each] = []  
  
parsingTable(rules)  
print(**"=========================================================================="**)  
print(**"\t\tParsing Table"**)  
**for** row **in** table:  
 print(row, table[row])  
print(**"=========================================================================="**)

**OUTPUT :**

