EXPERIMENT 07

CLASS: TE CMPN A ROLL NO. : 19
NAME: REBECCA DIAS PID: 182027

Aim: To find the first and follow sets of a given grammar.

Theory:

What are first and follow sets?

The construction of a predictive parser is aided by two functions associated with a grammar G. These functions, FIRST and FOLLOW, allow us to fill in the entries of a predictive parsing table for G.

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).

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)

How are these sets used by the parser?

To check if given grammar is LL(1) or not, we need to construct a parse table. This table can be constructed using FIRST and FOLLOW sets. Given below are the rules on how to construct the parse table using FIRST and FOLLOW sets –
For each production X →α

– for each terminal t in First(α): put α in table[X,t]

– if epsilon is in First(α) then:

for each terminal t in Follow(X): put α in table[X,t]

The grammar is not LL(1) if and only if there is more than one entry for any cell in the table.

Find the FIRST and FOLLOW of a grammar –

Rebecca Dias 19 Rebecca

Q

S → aBDh

B → cC

C → bC | E

D → EF

E → g | E

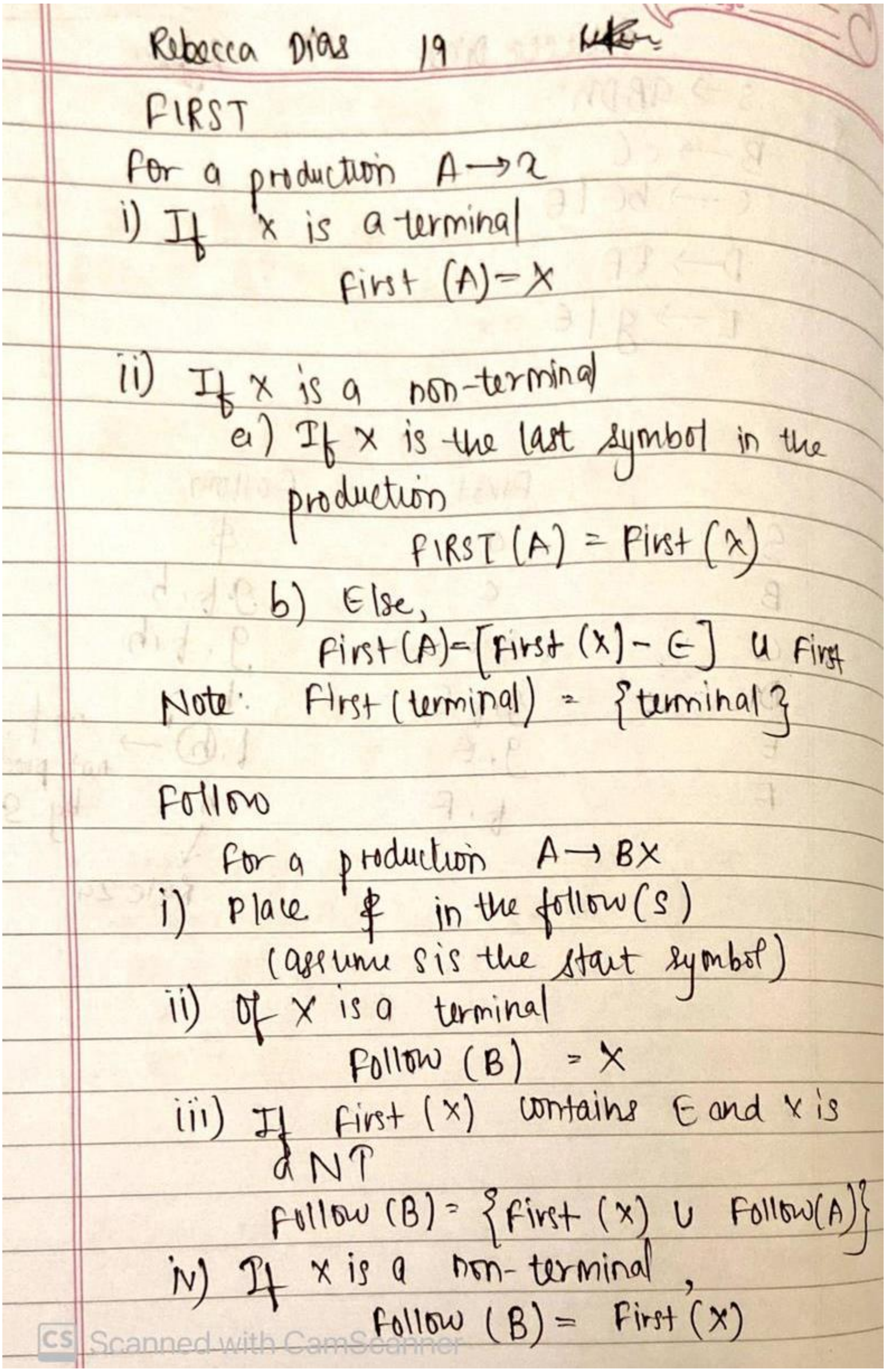
	First	Follow
S	a	$
B	c	g, f, h
C	b, E	g. f, h
D	g, f, E	h
E	g, E	f, h
F	f, F	h

→ no $, f, t not produce by S

↓ Rule 2a

Implementation:

Psuedo code



Rebecca Dias 19

FIRST

For a production $A \rightarrow \alpha$

i) If x is a terminal

First (A) = X

ii) If x is a non-terminal

a) If x is the last symbol in the production

FIRST (A) = First (x)

b) Else,

First (A) = [First (x) − ∈] ∪ First

Note: First (terminal) = {terminal}

Follow

For a production A → BX

i) Place $ in the follow (S)

(assume S is the start symbol)

ii) If X is a terminal

Follow (B) = X

iii) If First (X) contains ∈ and X is a NT

Follow (B) = {First (X) ∪ Follow (A)}

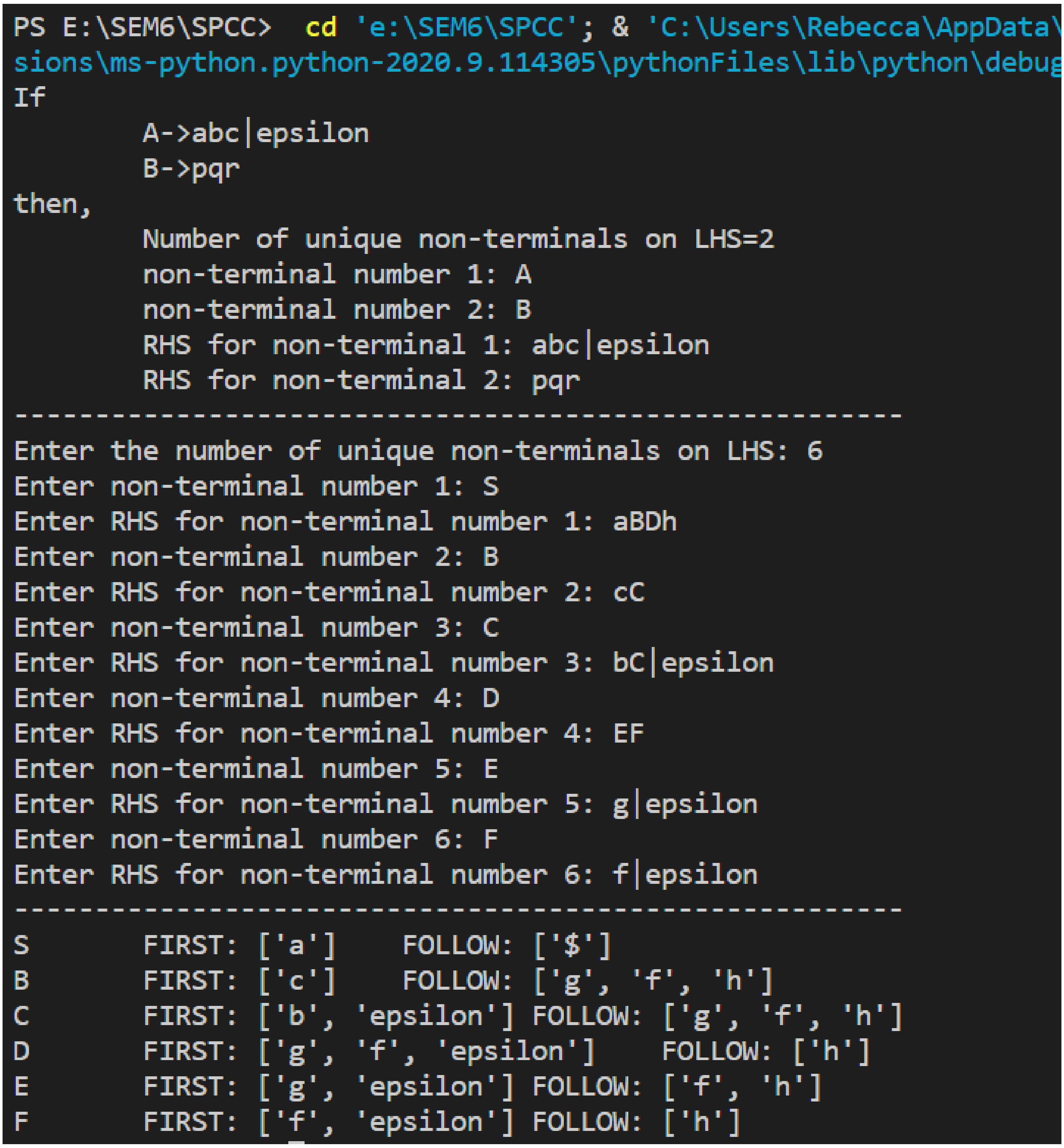
iv) If X is a non-terminal,

Follow (B) = First (X)

CODE:

```
print("If\n\tA->abc|epsilon\n\tB->pqr\nthen,\n\tNumber of unique non-terminals on LHS=2\n\tnon-terminal number 1: A\n\tnon-terminal number 2: B\n\tRHS for non-terminal 1: abc|epsilon\n\tRHS for non-terminal 2: pqr")
print("-"*55)
num=int(input("Enter the number of unique non-terminals on LHS: "))
nt_list=[]
production_list=[]
first_list=[]
follow_list=[]
for i in range(num):
    nt=input(f"Enter non-terminal number {i+1}: ")
    production=input(f"Enter RHS for non-terminal number {i+1}: ")
    nt_list.append(nt)
    production_list.append(production)
    first_list.append([])
    follow_list.append([])
follow_list[0].append('$')

def nt_present(f_list,production,index,j):
    for first_value in first_list[index]:
        if (first_value=="epsilon"):
            if (j==len(production)-1):
                f_list.append("epsilon")
            elif (production[j+1] in nt_list):
                nt_present(f_list,production,nt_list.index(production[j+1]),j+1)
            elif (production[j+1] not in nt_list):
                f_list.append(production[j+1])
        else:
            f_list.append(first_value)
    return


for i in reversed(range(num)):
    nt=nt_list[i]
    productions=production_list[i]
    t_productions=productions.split("|")
    j=0
    for production in t_productions:
        if (production=="epsilon"):
            first_list[i].append(production)
            continue
        elif (production[j] not in nt_list):
            first_list[i].append(production[j])
        else:
            index=nt_list.index(production[j])
            nt_present(first_list[i],production,index,j)

def follow_nt_present(p_idx,f_list,production,j):
    nt_index=nt_list.index(production[j])
    for first in first_list[nt_index]:
        if (first=="epsilon"):
            if (j==len(production)-1):
                for follow in follow_list[p_idx]:
                    f_list.append(follow)
            elif (production[j+1] not in nt_list):
                f_list.append(production[j+1])
            else:
                follow_nt_present(p_idx,f_list,production,j+1)
        else:
            f_list.append(first)
    return

for prod_idx,productions in enumerate(production_list):
    for production in productions.split("|"):
            if (production=="epsilon"):
                continue
            else:
                for j in range(len(production)):
                    if (production[j] in nt_list):
                        nt_index=nt_list.index(production[j])
                        if (j==len(production)-1):
                            if (nt_index==prod_idx):
                                continue
                            for follow_value in follow_list[prod_idx]:
                                follow_list[nt_index].append(follow_value)
                        elif (production[j+1] not in nt_list):
                            follow_list[nt_index].append(production[j+1])
                        else:
                            follow_nt_present(prod_idx,follow_list[nt_index],pro
duction,j+1)

print("-"*55)
for i in range(num):
    print(f"{nt_list[i]}\tFIRST: {first_list[i]}\tFOLLOW: {follow_list[i]}")
```

OUTPUT:



```
PS E:\SEM6\SPCC>  cd 'e:\SEM6\SPCC'; & 'C:\Users\Rebecca\AppData\
sions\ms-python.python-2020.9.114305\pythonFiles\lib\python\debug
If
        A->abc|epsilon
        B->pqr
then,
        Number of unique non-terminals on LHS=2
        non-terminal number 1: A
        non-terminal number 2: B
        RHS for non-terminal 1: abc|epsilon
        RHS for non-terminal 2: pqr
-------------------------------------------------------
Enter the number of unique non-terminals on LHS: 6
Enter non-terminal number 1: S
Enter RHS for non-terminal number 1: aBDh
Enter non-terminal number 2: B
Enter RHS for non-terminal number 2: cC
Enter non-terminal number 3: C
Enter RHS for non-terminal number 3: bC|epsilon
Enter non-terminal number 4: D
Enter RHS for non-terminal number 4: EF
Enter non-terminal number 5: E
Enter RHS for non-terminal number 5: g|epsilon
Enter non-terminal number 6: F
Enter RHS for non-terminal number 6: f|epsilon
-------------------------------------------------------
S       FIRST: ['a']    FOLLOW: ['$']
B       FIRST: ['c']    FOLLOW: ['g', 'f', 'h']
C       FIRST: ['b', 'epsilon'] FOLLOW: ['g', 'f', 'h']
D       FIRST: ['g', 'f', 'epsilon']    FOLLOW: ['h']
E       FIRST: ['g', 'epsilon'] FOLLOW: ['f', 'h']
F       FIRST: ['f', 'epsilon'] FOLLOW: ['h']
```

CONCLUSION:

The first and follow sets can be found for any given grammar.