6: Find First Set of given grammar:

Input:

E->TX

X->+TX| ε

T->FY

Y->*FY| ε

F->(E) | z

Expected Output:

First (E)= [(,z]

First(X)= [+,ε]

First(T)= [(,z]

First(Y)= [*,ε]

First(F)= [(,z]

Theory/Logic:

FIRST:

FIRST is applied to the right hand side of a production rule, and tells us all the terminal symbols that can start sentences derived from that right hand side. It is defined as:

1. For any terminal symbol a, FIRST (a)={a} Also FIRST (ε)={ε }

2. For any non-terminal A with production rules A→α1|α2|….∨αn,FIRST (A)=FIRST (α1)∪FIRST (α2)∪…∪FIRST (αn).

3. For any right hand side of the form:β1 β2…βn (where each βi is a terminal or a nonterminal) we have:

1. FIRST(β1) is in FIRST(β1 β2…βn) if β1 can derive ε, then FIRST(β1), FIRST(β2) is also in FIRST(β1 β2…βn) if both β1 and β2 can derive ∈, then FIRST (β3) is also in FIRST(β1 β2…βn)

2. if β1 β2…βi can derive ε, then FIRST(βi+1) is also in FIRST(β1

β2…βn) 3. ε is in FIRST(β1 β2…βn) only if ε is in FIRST(βi), for all

0≤ i≤ n

FIRST can be applied to any r.h.s., and returns a set of terminal symbols. Thus, if X is the current non-terminal (ie. leftmost in the current sentential form), and a is the next symbol on the input,
then we want to look at the r.h.s. of the production rules for X and choose the one whose FIRST set contains a.

Why FIRST?

If the compiler would have come to know in advance, that what is the "first character of the string produced when a production rule is applied", and comparing it to the current character or token in the input string it sees, it can wisely take decision on which production rule to apply.

Let's take the example grammar:

S→cAd

A→bc|a

And the input string is "cad".
Thus, in the example above, if it knew that after reading character 'c' in the input string and applying S→cAd, next character in the input string is 'a', then it would have ignored the production rule A→bc (because 'b' is the first character of the string produced by this production rule, not 'a'), and directly use the production rule A→a(because 'a' is the first character of the string produced by this production rule, and is same as the current character of the input string which is also 'a'). Hence it is validated that if the compiler/parser knows about first character of the string that can be obtained by applying a production rule, then it can wisely apply the correct production rule to get the correct syntax tree for the given input string.

Source Code:

```
fterminals=[]
def pterminals(chars):
    global fterminals
    fterminals.append(chars)

def getterminal(cha):
    global diction
    global non_terminals
    att=''
    a=diction[cha]
    if a[0] in non_terminals:
        getterminal(a[0])
    else:
        if '|' in a:
            ind1=a.index('|')
            att=a[0]+a[ind1+1:]
        else:
            att=a[0]
        return att


t=int(input("Enter the total no. of grammar: "))
gra=[]
temp=''
for i in range(t):
    temp=input(f"Enter the elements of {i+1} grammar: ")
    gra.append(temp)
    temp=''

non_terminals=[]
for i in gra:
    temp=i[0]
    non_terminals.append(temp)
    temp=''

diction={}

for i in range(len(gra)):
    diction[non_terminals[i]]=gra[i][3:]

tstr=''
```

```
for i in range(len(gra)):
    if gra[i][3] not in non_terminals:
        tstr=gra[i][3]
        if '|' in gra[i]:
            ind=gra[i].index('|')
            tstr+=gra[i][ind+1:]
            pterminals(tstr)
        else:
            pterminals(tstr)
    else:
        aa=getterminal(gra[i][3])
        pterminals(aa)

print('\n\n')
for i in range(len(fterminals)):
    print(f'First({non_terminals[i]}) -> {fterminals[i]}')
```

OUTPUT:

```
In [91]: runfile('C:/Users/Admin/study material/sem-7/Practical/CD/Practical-5/untitled0.py',
wdir='C:/Users/Admin/study material/sem-7/Practical/CD/Practical-5')

Enter the total no. of grammar: 5

Enter the elements of 1 grammar: E->FX

Enter the elements of 2 grammar: X->+TX|e

Enter the elements of 3 grammar: T->FY

Enter the elements of 4 grammar: Y->*FY|e

Enter the elements of 5 grammar: F->(E)|z



First(E) -> (z
First(X) -> +e
First(T) -> (z
First(Y) -> *e
First(F) -> (z

In [92]:
```