**Name**: Shubham Lad

**Class**: MSc Computer Science (Part 1)

**Roll No**: 512

**Semester**: II

**Subject**: Design and Implementation of Modern Compilers

**Topic**: First & Follow

**Aim**: Python code for finding First and Follow

**Introduction:**

FIRST and FOLLOW are two functions associated with grammar that help us fill in the entries of an M-table.

FIRST(): It is a function that gives the set of terminals that begin the strings derived from the production rule.

A symbol c is in FIRST (α) if and only if α ⇒ cβ for some sequence β of grammar symbols.

A terminal symbol a is in FOLLOW (N) if and only if there is a derivation from the start symbol S of the grammar such that S ⇒ αNαβ, where α and β are a (possible empty) sequence of grammar symbols. In other words, a terminal c is in FOLLOW (N) if c can follow N at some point in a derivation.

**Benefit of FIRST( ) and FOLLOW( )**

* It can be used to prove the LL (K) characteristic of grammar.
* It can be used to promote in the construction of predictive parsing tables.
* It provides selection information for recursive descent parsers.

**Computation of *FIRST***

*FIRST (α) is defined as the collection of terminal symbols which are the first letters of strings derived from α.*

FIRST (α) = {α |α →∗ αβ for some string β}

If X is Grammar Symbol, then First (X) will be:

* If X is a terminal symbol, then FIRST(X) = {X}
* If X → ε, then FIRST(X) = {ε}
* If X is non-terminal & X → a α, then FIRST (X) = {a}
* If X → Y1, Y2, Y3, then FIRST(X) will be:
  1. If Y is terminal, then FIRST(X)=FIRST (Y1,Y2,Y3)={Y1}
  2. If Y1 is non-terminal and if Y1 does not derive to an empty string i.e.,

if FIRST(Y1) does not contain ε then, FIRST(X)=FIRST(Y1,Y2,Y3)=FIRST(Y1)

* 1. If FIRST (Y1) contains ε, then, FIRST(X)=FIRST(Y1,Y2,Y3)=FIRST(Y1)−{ε}∪ FIRST(Y2, Y3)

Similarly, FIRST (Y2, Y3) = {Y2}, If Y2 is terminal otherwise if Y2 is non-terminal then

* FIRST (Y2, Y3) = FIRST (Y2), if FIRST (Y2) does not contain ε.
* If FIRST (Y2) contain ε, then FIRST (Y2, Y3) = FIRST (Y2) − {ε} ∪ FIRST (Y3)

**Computation of *FOLLOW***

*Follow (A) is defined as the collection of terminal symbols that occur directly to the right of A.*

FOLLOW(A) = {a|S ⇒\* αAaβ where α, β can be any strings}

* If S is the start symbol, FOLLOW (S) ={$}
* If production is of form A → α B β, β ≠ ε.

1. If FIRST (β) does not contain ε then, FOLLOW (B) = {FIRST (β)}

Or

1. If FIRST (β) contains ε (i. e. , β ⇒\* ε), then

FOLLOW (B) = FIRST (β) − {ε} ∪ FOLLOW (A)

ie. when β derives ε, then terminal after A will follow B.

* If production is of form A → αB, then Follow (B) ={FOLLOW (A)}.

**Code:**

import sys

sys.setrecursionlimit(60)

def first(string):

    #print("first({})".format(string))

    first\_ = set()

    if string in non\_terminals:

        alternatives = productions\_dict[string]

        for alternative in alternatives:

            first\_2 = first(alternative)

            first\_ = first\_ |first\_2

    elif string in terminals:

        first\_ = {string}

    elif string=='' or string=='@':

        first\_ = {'@'}

    else:

        first\_2 = first(string[0])

        if '@' in first\_2:

            i = 1

            while '@' in first\_2:

                #print("inside while")

                first\_ = first\_ | (first\_2 - {'@'})

                #print('string[i:]=', string[i:])

                if string[i:] in terminals:

                    first\_ = first\_ | {string[i:]}

                    break

                elif string[i:] == '':

                    first\_ = first\_ | {'@'}

                    break

                first\_2 = first(string[i:])

                first\_ = first\_ | first\_2 - {'@'}

                i += 1

        else:

            first\_ = first\_ | first\_2

    #print("returning for first({})".format(string),first\_)

    return  first\_

def follow(nT):

    #print("inside follow({})".format(nT))

    follow\_ = set()

    #print("FOLLOW", FOLLOW)

    prods = productions\_dict.items()

    if nT==starting\_symbol:

        follow\_ = follow\_ | {'$'}

    for nt,rhs in prods:

        #print("nt to rhs", nt,rhs)

        for alt in rhs:

            for char in alt:

                if char==nT:

                    following\_str = alt[alt.index(char) + 1:]

                    if following\_str=='':

                        if nt==nT:

                            continue

                        else:

                            follow\_ = follow\_ | follow(nt)

                    else:

                        follow\_2 = first(following\_str)

                        if '@' in follow\_2:

                            follow\_ = follow\_ | follow\_2-{'@'}

                            follow\_ = follow\_ | follow(nt)

                        else:

                            follow\_ = follow\_ | follow\_2

    #print("returning for follow({})".format(nT),follow\_)

    return follow\_

no\_of\_terminals=int(input("Enter no. of terminals: "))

terminals = []

print("Enter the terminals :")

for \_ in range(no\_of\_terminals):

    terminals.append(input())

no\_of\_non\_terminals=int(input("Enter no. of non terminals: "))

non\_terminals = []

print("Enter the non terminals :")

for \_ in range(no\_of\_non\_terminals):

    non\_terminals.append(input())

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

no\_of\_productions = int(input("Enter no of productions: "))

productions = []

print("Enter the productions:")

for \_ in range(no\_of\_productions):

    productions.append(input())

#print("terminals", terminals)

#print("non terminals", non\_terminals)

#print("productions",productions)

productions\_dict = {}

for nT in non\_terminals:

    productions\_dict[nT] = []

#print("productions\_dict",productions\_dict)

for production in productions:

    nonterm\_to\_prod = production.split("->")

    alternatives = nonterm\_to\_prod[1].split("/")

    for alternative in alternatives:

        productions\_dict[nonterm\_to\_prod[0]].append(alternative)

#print("productions\_dict",productions\_dict)

#print("nonterm\_to\_prod",nonterm\_to\_prod)

#print("alternatives",alternatives)

FIRST = {}

FOLLOW = {}

for non\_terminal in non\_terminals:

    FIRST[non\_terminal] = set()

for non\_terminal in non\_terminals:

    FOLLOW[non\_terminal] = set()

#print("FIRST",FIRST)

for non\_terminal in non\_terminals:

    FIRST[non\_terminal] = FIRST[non\_terminal] | first(non\_terminal)

#print("FIRST",FIRST)

FOLLOW[starting\_symbol] = FOLLOW[starting\_symbol] | {'$'}

for non\_terminal in non\_terminals:

    FOLLOW[non\_terminal] = FOLLOW[non\_terminal] | follow(non\_terminal)

#print("FOLLOW", FOLLOW)

print("{: ^20}{: ^20}{: ^20}".format('Non Terminals','First','Follow'))

for non\_terminal in non\_terminals:

    print("{: ^20}{: ^20}{: ^20}".format(non\_terminal,str(FIRST[non\_terminal]),str(FOLLOW[non\_terminal])))

**Output:**

